VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

KOEVOLUCE KARTÉZSKÝCH GENETICKÝCH ALGORITMŮ A NEURONOVÝCH SÍTÍ

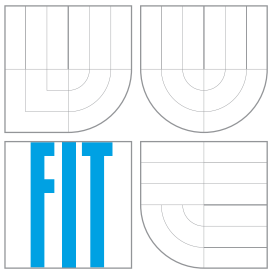
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. ADAM KOLÁŘ

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

KOEVOLUCE KARTÉZSKÝCH GENETICKÝCH ALGORITMŮ A NEURONOVÝCH SÍTÍ

COEVOLUTION OF CARTESIAN GENETIC

ALGORITHMS AND NEURAL NETWORKS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. ADAM KOLÁŘ

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Ing. FRANTIŠEK ZBOŘIL, CSc.

BRNO 2014

Abstrakt

Cílem práce bylo ověřit možnost spojení metod kartézského genetického programování a neuronových sítí. Výsledek práce reprezentuje sada experimentů s úlohami vyvažování tyče, průchodu agenta bludištěm a detekce buňek rakoviny prsu, provedených za pomocí implementované knihovny. Použil jsem jak nepřímé, tak přímé zakódování dopředné i rekurentní sítě. Zaměřil jsem se na nalezení nejlepší konfigurace výpočtu, detekci přeučování jedinců během evoluce, míru robustnosti řešení a možnosti stimulace řešení pomocí fitness funkce. Zjistil jsem, že obecně nejlépe konvergují řešení s nižšími hodnotami parametru n_c a n_r a jsou také méně náchylné k přeučování. Úpravami fitness funkce jsem evolvoval kontrolér, který redukoval rozkmit vyvažované tyče. Klasifikátor buňek rakoviny rozpoznával korektně více než 98% vzorků, čímž překonal většinu srovnávaných metod. Podařilo se také navrhnout model bludiště, ve kterém agent úspěšně plnil vícekrokové úlohy.

Abstract

The aim of the thesis is to verify synergy of genetic programming and neural networks. Solution is provided by set of experiments with implemented library built upon benchmark tasks. I've done experiments with directly and also indirectly encoded neural netwrok. I focused on finding robust solutions and the best calculation of configurations, overfitting detection and advanced stimulations of solution with fitness function. Generally better solutions were found using lower values of parameters n_c and n_r. These solutions tended less to be overfitted. I was able to evolve neurocontroller eliminating oscilations in pole balancing problem. In cancer detection problem, precision of provided solution was over 98%, which overcame compared techniques. I succeeded also in designing of maze model, where agent was able to perform multistep tasks.

Klíčová slova

Backpropagation, Neuron, Chromozom, Kartézské genetické programování, Koevoluce, Developmental network, Nepřímé zakódování sítě, Přímé zakódování sítě, Jordanova síť, Stavová jednotka, Wampus problém, Problém průchodu bludištěm, Problém vyvažování tyče, Wisconsinský datový set, Detekce rakoviny prsu

Keywords

Backpropagation, Neuron, Chromozom, Cartesian genetic programming, Coevolution, Developmental network, Indirect encoding of the net, Direct encoding of the net, Jordan network, State unit, Wampus problem, Maze go through problem, Pole balancing problem, Wisconsin dataset, Breast cancer detection

Citace

Adam Kolář: Koevoluce kartézských genetických
algoritmů a neuronových sítí, diplomová práce, Brno, FIT VUT v Brně, 2014

Koevoluce kartézských genetických algoritmů a neuronových sítí

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Doc. Ing. Františka V. Zbořila, CSc. ve spolupráci s Ing. Michaelou Šikulovou.

.....
Adam Kolář
27. května 2014

Poděkování

Tímto bych chtěl poděkovat svému vedoucímu Doc. Ing. Františku V. Zbořilovi, CSc. za cenné rady při psaní diplomové práce, stejně tak Ing. Michaele Šikulové za odborné konzultace v oblasti kartézského genetického programování.

Obsah

Kapitola 1

Úvod

Soudobé informační technologie jsou často nuceny analyzovat z pohledu složitosti $\mathcal{NP}$ úplné problémy. Jedná se o úlohy, jenž řeší nedeterministický *Turingův stroj* v polynomiálním čase a na které jsou polynomiálně redukovatelné všechny ostatní $\mathcal{NP}$ problémy. Využití běžných deterministických algoritmů zde selhává, zejména díky enormnímu stavovému prostoru zadaných problémů. Oblast *softcomputing* přináší několik použitelných postupů pro řešení tohoto druhu problémů. Mezi ně patří metody evolučních algoritmů, neuronových sítí, fuzzy logiky a teorie chaosu. Každá z popsaných skupin je vhodnější pro jinou sadu úloh. V této práci se zaměříme na evoluční algoritmy a neuronové sítě.

Neuronové sítě jsou obvykle reprezentovány množinou uzlů (neuronů) a propojovacích hran s příslušnými vahami. Topologie takové neuronové sítě bývá často statická, vzniká empiricky a pro správnou aproximaci řešení je v některých případech třeba mít trénovací množinu vstupů a výstupů, na jejímž základě upravujeme hodnoty vah, které rozhodnou o aktivitě neuronů uvnitř sítě. V případě složitých topologií vznikají problémy se správnou konvergencí hodnot vah v průběhu učení sítě, s uvíznutími v lokálních minimech řešení atp.. Neexistuje žádný univerzální způsob, jak správně vytvořit a natrénovat aplikačně specifickou neuronovou síť. Položme si ovšem otázku, jestli by i samotný způsob tvorby struktury sítě a její učení nemohlo být výsledkem evolučního procesu.

Tuto alternativu pokrývá koevoluce genetického programování a neuronových sítí. Snažíme se zakódovat buď nepřímo mechanizmy budující síť s požadovanými vlastnostmi, případě přímo topologii sítě i s jejími vahami do genotypu. Během generací se řešení zpřesňuje a genotyp podléhá evolučnímu procesu. V každé generaci se genotyp transformuje do podoby výsledného fenotypu, kterým je vytrénovaná obecná neuronová síť. Ta se dále vyhodnocuje. Představený koncept je možným východiskem pro návrh topologie a vah sítě u složitějších dynamických problémů.

V této práci bude nastíněn teoretický základ pro evoluční algoritmy *(kap. 2)* (především genetické programování *(kap. 2.3)*) a neuronové sítě *(kap. 3)*. V části 4 bude detailně rozebrána koevoluce genetického programování a neuronových sítí. Zaměříme se zejména na způsob zakódování sítě, ohodnocení řešení a způsob propojení obou popsaných technik. V kapitole 4.1 se budeme věnovat nepřímé alternativě zakódovaní neuronové sítě s využitím *Developmental Network*. V kapitole 4.2 se pak budeme věnovat alternativě přímého zakódování. U přímého zakódování popíšeme mechanizmy zakódování dopředné neuronové sítě *(kap. 4.2.2)*, stejně jako rekurentní varianty sítě vycházející z *Jordanova* návrhu *(kap. 4.2.3)*. Kapitola 5 se zabývá návrhem knihovny určené pro experimentování s předloženými návrhy koevoluce neuronových sítí a genetického programování. S knihovnou budou provedeny experimenty nad vybranými úlohami *(kap. 6)*. Našim cílem je otestovat jak složitější

problémy typu průchod agenta bludištěm *(kap. 6.3)*, které pro určité alternativy zakódování doposud nebyly nikdy prezentovány, tak prakticky zaměřené testbanch úlohy. Mezi ně patří problém vyvažování tyče na pohyblivé podložce *(kap. 6.2)* nebo úloha klasifikace buněk rakoviny prsu *(kap. 6.1)*. Experimenty slouží nejenom ke srovnání koevolučních technik se stávajícími přístupy, ale také k popisu vlastností evolvovaných sítí (robustnost, míra přeučování, možnosti ovlivnění vlastností řešení stimulací *fitness funkce*) a nalezení optimálních parametrů nastavení konfigurace výpočtů. V závěru práce *(kap. 7)* provedeme globální zhodnocení experimentů a dosažených výsledků.

Kapitola 2

Evoluční algoritmy

Termín evoluční algoritmy [3] do sebe zahrnuje několik metod řešení problémů, pro které je prostor řešení příliš velký, má mnoho extrémů či nelze popsat pomocí matematické analýzy. Spojitost mezi metodami je dána jejich podstatou, jejich přímou souvislostí s teoriemi *Charlese Darwina* o přirozeném výběru, transmutaci druhů a vlivu používání a nečinnosti, kdy kandidátními řešeními evoluce jsou jedinci populace. Evoluční algoritmy při řešení využívají jen několik základních faktů, kterými *Jean Baptiste de Lamarck*, *Charles Darwin* a konečně *Johan Gregor Mendel* popisovali průběh evoluce druhů na Zemi.

- `Přirozený výběr` - Silnější jedinci populace (dle hodnotícího kritéria) se s vyšší pravděpodobností účastní na reprodukci. Tito jedinci mají i vyšší šanci přežít déle než jedinci slabší.
- `Genetický drift` - Náhodná událost v životě jedince, jenž ovlivňuje celou populaci. Případ driftu je genetická mutace či předčasné úmrtí silného jedince. Mutace může do populace vnést nový fenotyp, který je diametrálně odlišný od typického jedince v populaci.
- `Variabilita` - Populace je geneticky rozmanitá bez ohledu na prostředí.
- `Reprodukce` - Reprodukční proces je ovlivněn přirozeným výběrem, což umožní šíření kvalitních vlastností dalšími generacemi. Při reprodukci rodiče náhodně předávají potomku části své genetické informace.

U evolučních stochastických optimalizačních metod je nutné se vypořádat s několika problémy. Především se zakódováním jedinců, způsobem jejich ohodnocení, technikou mutací a křížení, velikostí populace a časem ukončení celého algoritmu. Stěžejní je také dodat dostatek specifických informací, vztahujících se ke konkrétnímu problému. To plyne z *'No Free Lunch' teorému (dále NFL teorém).*

Teorém 2.1. *`NFL teorém` - Žádný prohledávací algoritmus, který navštíví veškeré stavy optimalizační úlohy nejvýše jednou, nemůže být v průměru efektivnější, než systematické prohledávání všech možností. V průměru je tedy náhodné prohledávání stavového prostoru stejně dobré jako použití optimalizačních metod. To však neplatí pro konkrétní úlohu, u které lze zajistit expertní bázi znalostí.*

2.1 Základní pojmy

Veškeré aplikačně specifické informace vztahující se k definovaným pojmům najdeme v uvedených kapitolách, týkajících se navrhované struktury koevoluce genetického programování a neuronových sítí. Pojmy jsou proto definovány co nejobecněji tak, aby byl pokryt jejich význam u všech základních metod evolučních algoritmů [1].

Definice 2.2. ***Genotyp*** *- Genotyp je množina genů G populace P, její kompletní genetický materiál. Mějme $G = \{g_1, g_2, .., g_n\}$, $n \in \mathbb{N}$. Konkrétní variace genů genotypu je zobrazena do chromozomu, který představuje konkrétní zakódování problému.*

Definice 2.3. ***Fenotyp*** *- Výsledná kombinace genotypu a prostředí, ve kterém se genotyp nachází. V našem případě se jedná o požadovaný produkt, tedy interpretaci formální reprezentace chromozomu. U jednodušších problémů je časté přímé zobrazení mezi genotypem a fenotypem. Není tomu tak ale vždy.*

Definice 2.4. ***Populace a chromozom*** *- Populace P v generaci t je reprezentována množinou jedinců α_i, $i \in \mathbb{N}$, existujících v generaci t, $P(t) = \{\alpha_1, \alpha_2, .., \alpha i, .., \alpha_n\}$, $n \in \mathbb{N}$. Jedince, neboli chromozom, chápeme obecně jako variaci genotypu s opakováním libovolné délky, $\alpha_i \in \overline{V}_n^{|G|}$, $n \in \mathbb{N}$. Množina všech variací nad G je $V(G) = \left\{\overline{V}_n^{|G|} \mid n \in \mathbb{N}\right\}$. Libovolná populace je podmnožinou potenční množiny všech takových variací nad G, $P(t) \subseteq \mathcal{P}(V(G))$. Chromozom může mít formu nejenom symbolického řetěze (genetické algoritmy), ale i spustitelné struktury (genetické programování)*

Definice 2.5. ***Fitness funkce*** *- Jedná se o ohodnocení kvality jedince v populaci, které rozhoduje jak o četnosti účasti na předávání genetického materiálu, tak na délce jedincova přežití. Fitness $f : \alpha \to \mathcal{R}^+$*

Definice 2.6. ***Mutace*** *- Unární operátor nad genotypem G $M : \alpha \to \beta$, $\alpha, \beta \in V(G)$, který způsobí stochastickou změnu hodnoty genu v chromozomu či změnu struktury chromozomu. Jeho účelem je vytvářet mutanty, jedince z hlediska výsledků operátoru křížení netypické pro populaci. Je to z toho důvodu, abychom nevhodným nastavením parametrů křížení nekonvergovali do množiny řešení v lokálním minimu. Obvykle je použita jako doplněk operátoru křížení, ale genetické programování ji používá ve svých alternativách i jako hlavní mechanizmus pro hledání správného řešení.*

Definice 2.7. ***Křížení*** *- Binární operátor $C : \alpha \times \beta \to \gamma$, $\alpha \in V(G_1)$, $\beta \in V(G_2)$ a $\gamma \in \{V(G_1 \cup G_2)\}$, jenž zkombinuje dva chromozomy do nového potomka. Cílem je podpořit a předat do další populace kvalitní část genotypu. O šíření kvalitního genotypu populacemi u genetického programování pojednává schema teorém (viz. teorém 2.10).*

Definice 2.8. ***Operátor výběru jedince z populace*** *- Mějme populaci P a jedince α, $\alpha \in P$. Potom definujme selekční operátor S, $S : P \to \alpha$. Cílem operátoru je vhodně vybírat jedince pro mutace a křížení. Výběr je podmíněn pravděpodobnostním rozložením, které definujeme pomocí fitness funkce jedinců v populaci. Mezi základní formy selekce patří ruletový výběr (obecně spadající do metod proporcionálního výběru). Tyto metody neeliminují crowding (definice 2.13). Pro potlačení vlivu jedinců, jejichž hodnota fitness výrazně převyšuje hodnotu fitness zbytku populace, jsou použity metody turnaje nebo selekce. Selekce je založená na uspořádání jedinců dle velikosti jejich fitness a následném výběru s daným*

[1] Pojmy neodpovídají přesným biologickým definicím a jsou upraveny pro naše účely.

rozložením pravděpodobnosti (exponenciální, lineární). Umožňuje měnit velikost selekčního tlaku (viz. definice 2.9). Metoda turnaje vybírá jednotlivé jedince do nové populace tak, že z náhodně zvolené podmnožiny jedinců populace vždy vybere nejlépe ohodnoceného jedince do populace nové [8].

Definice 2.9. *`Selekční tlak` - Při výběru jedince pro operace mutace a křížení musí operátor selekce zohledňovat jednak pravděpodobnostní model výběru nad populací ovlivněný fitness funkcí, zároveň musí zajistit dostatečnou různorodost vybíraných jedinců, aby nedošlo k přílišnému upřednostňování určité části populace. Selekční tlak definuje míru zohledňování fitness funkce pro výběr jedince. Vyšší selekční tlak implikuje větší upřednostňování jedinců s vyšší fitness funkcí a obecně urychluje konvergenci algoritmu. Nezaručuje ovšem konvergenci do globálního minima. Čas nutný pro unifikaci populace nejsilnějším jedincem pouze pomocí selekčního algoritmu nazvěme takeover time.*

Velikost selekčního tlaku S_p definujeme jako $S_p = \frac{M_b - M_a}{\sigma}$, kde M_a je průměrná hodnota fitness před provedením selekce a M_b průměrná hodnota po obnovení populace po selekci. Parametr σ pak definuje rozptyl fitness před selekcí.

Teorém 2.10. *`Schema teorém` - Jedná se o matematický popis šíření genotypu populací, definovaný pro genetické algoritmy. Schema teorém existuje ve své podobně i pro genetické programování. Je uveden důkaz schema teorému publikovaného Johnem Hollandem pro genetické algoritmy [4].*

Důkaz. Uvažujme genotyp $G = \{0, 1, *\}$ a schema $H \in G^n$. Hodnoty 0 a 1 jsou pevně dány. Symbol $*$ reprezentuje jednu z alternativ $\{0, 1\}$. Pokud v H nahradíme $*$ konkrétními symboly, vzniká vzor schematu. Každé H tedy reprezentuje podmnožinu chromozomů pro daný genotyp.

Řád schematu $O(H)$ určuje počet pevně daných pozic H, délka schematu $\delta(H)$ vzdálenost mezi první a poslední pevnou pozicí schematu. Mějme populaci $P(t)$ a schema H. Tomuto schematu odpovídá x chromozomů v populaci v čase t, $x = m(H, t)$. Uvažujeme-li průměrnou fitness hodnotu schematu H $f(H)$ a průměrnou hodnotu fitness populace jako $\overline{f}$, pak

$$m(H, t+1) = m(H, t)\frac{f(H)}{\overline{f}}$$

Pokud je schema H kvalitní a jeho hodnota fitness převyšuje hodnotu průměrné fitness tak, že $f(H) = \overline{f} + c\overline{f}$, kde c je kladná konstanta, potom je po proběhnutí e generací

$$m(H, t+1) = m(H, t)(1 + c)^e$$

Počet jedinců se schematem H se pak bude dále exponenciálně zvětšovat. Tento jev nastane v případě, že schema nebude narušeno mutací s pravděpodobností p_m či křížením s pravděpodobností p_c. Nyní uvažujme jevy mutace a křížení a jejich vliv na udržení schematu v populaci. Cílem je, aby potomci chromozomů založených na H byli stále pokryti tímto schematem.

- `Mutace` - Pravděpodobnost, že schema bude poškozeno mutací je $p_{mcrash} = 1 - (1 - p_m)^{O(H)}$. Vyjadřujeme, že alespoň jedna z pevných pozic bude změněna.
- `Křížení` - Pravděpodobnost narušení schematu křížením je $p_{ccrash} = p_c \frac{\delta(H)}{n-1}$.

Celková pravděpodobnost zničení schematu $p_{crash} = p_{mcrash} + p_{ccrash}$. Pokud budeme uvažovat pravděpodobnost zničení schematu, počty jedinců se zvoleným schematem v další generaci budou

$$m(H, t+1) = m(H, t)\frac{f(H)}{\overline{f}}(1 - p_{crash})$$

$$m(H, t+1) = m(H, t)\frac{f(H)}{\overline{f}}\left((1 - p_m)^{O(H)} - p_c\frac{\delta(H)}{n-1}\right)$$

$$m(H, t+1) = m(H, t)(1 + c)\left((1 - p_m)^{O(H)} - p_c\frac{\delta(H)}{n-1}\right) \tag{2.1}$$

Z rovnice 2.1 vyplývají vlastnosti schematu H, které umožní, že bude v průběhu evoluce upřednostňováno a počet jeho výskytů bude během generací růst. Fitness hodnota takového schematu bude nadprůměrná, což zajišťuje konstanta $c > 0$. Abychom maximalizovali $m(H, t+1)$, hledáme krátká schemata, tj. s nízkým $\delta(H)$, nízkého řádu $O(H)$. Takové H nazýváme stavebními bloky a jejich zpracováním dostaneme konečné řešení. Příkladem, kdy teorie schematu neplatí, je oblast klamných problémů, která nás dovádí opětovně k potřebě mutací. □

2.2 Obecný průběh algoritmu

Nejprve je náhodně inicializována počáteční populace, tedy možná řešení zadané úlohy zakódovaná do chromozomů. Ty jsou ohodnoceny fitness funkcí f. Pravidlem je, že kvalitnější řešení mají tuto hodnotu vyšší a každou další iterací (následující generací) algoritmu by se průměrná hodnota fitness měla zvyšovat či zůstávat stejná. Na základě ohodnocení jsou vybíráni jedinci, na něž se aplikují operátory mutací a křížení.

Výběr operátorů je podmíněn zadanou pravděpodobností, ale není deterministický. Je to jedna z technik, jak při hledání extrémů funkce neuvíznout v lokálních minimech. Aplikací operátorů vznikají potomci. Ze stávající populace a potomků se vyberou nejvhodnější jedinci a vzniká populace nová.

Pro obnovu populace se používá několik strategií, obecně zapisovaných (μ, λ) nebo $(\mu + \lambda)$ podle toho, zda chceme vybrat nových μ rodičů pouze z potomků λ nebo z rodičů i potomků dohromady. Nová populace se opět ohodnotí a cyklus se opakuje.

Algoritmus nemá žádný explicitní konec, často však chod ukončujeme po pevném počtu iterací nebo při dostatečně malých rozdílech mezi ohodnoceními nejlepších jedinců populací v následujících generacích. Suboptimální řešení nese nejsilnější jedinec v poslední generaci.

```
1 inicializuj čas t
2 inicializuj novou populaci P(t)
3 ohodnoť populaci P(t)

4 while není splněna definovaná ukončující podmínka algoritmu:
5   t' = t + 1
6   Q(t) = vytvoř nové jedince z P(t) za pomoci evolučních operátorů
7   P(t') = vyber nejvhodnější jedince do nové generace z Q(t) a P(t)
8   ohodnoť populaci P(t')
9   t = t'
10 end while
11 return nejsilnější jedinec z P(t)
```

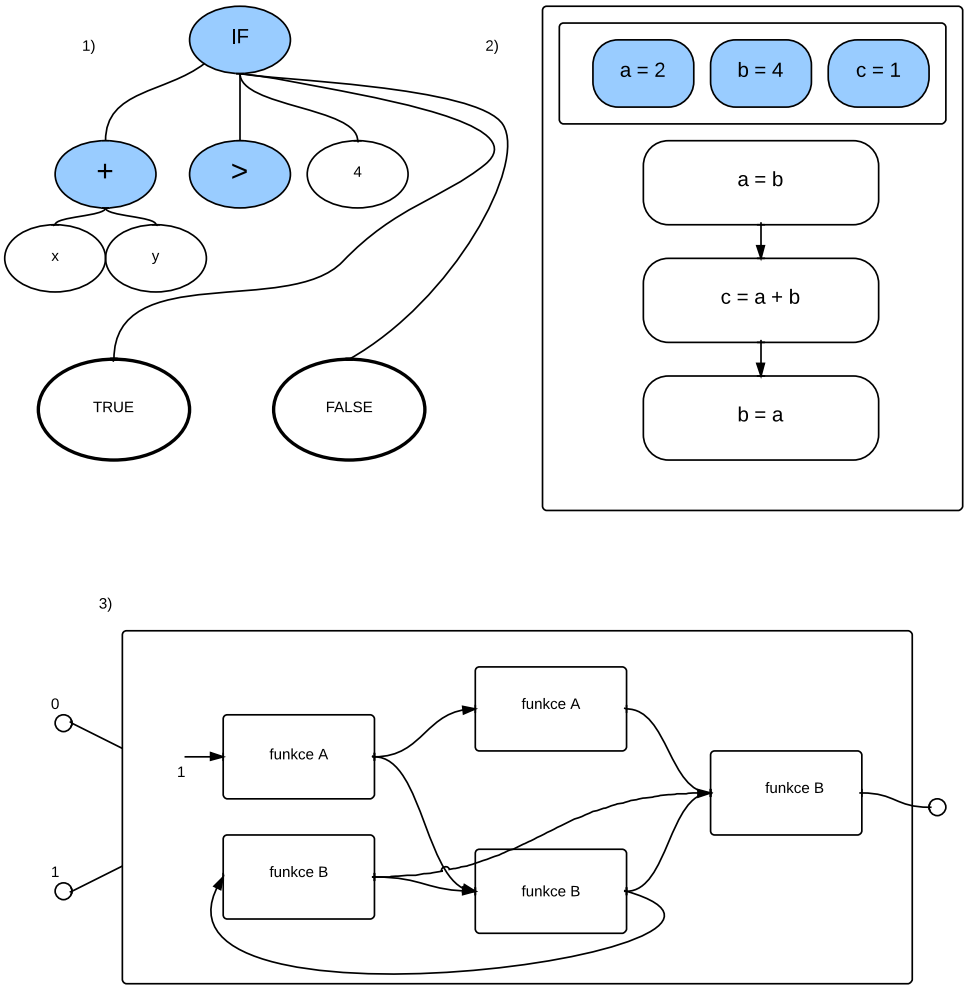
Algoritmus 1: Obecný průběh evolučního algoritmu.

2.3 Genetické programování

V 80.letech 20.století *John Koza* položil základy technice nazvané genetické programování [13], sloužící pro automatizovaný návrh počítačových programů. Hlavním rozdílem oproti genetickým algoritmům a jím příbuzných metodám je způsob zakódování jedinců, aplikování genetických operátorů na chromozomy a vyhodnocení jejich fitness. Průběh algoritmu se shoduje s obecnou podobou *(alg. 1)*.

2.3.1 Reprezentace chromozomu

Jedinci (chromozomy) populace jsou reprezentování spustitelnými programy. Během evoluce se tak vyvíjejí spustitelné struktury proměnlivé délky. Geny chromozomu mohou být nejenom terminály, ale i samostatné funkční jednotky. Struktury lze reprezentovat pomocí stromu, lineární posloupnosti instrukcí nebo obecného či acyklického orientovaného grafu (obr. 2.1), například u kartézského genetického programování *(kap. 2.4.1)*.



Obrázek 2.1: *(1) Znázornění možné topologie stromu pro rozhodovací problém* ***if else****. Jsou použity dvě proměnné, které budou inicializovány ze standardního vstupu. (2) Lineární posloupnost. Proměnné jsou inicializovány a poté se s nimi sekvenčně pracuje. (3) Obecné schéma orientovaného grafu. Systém má dva vstupy a jeden výstup. Jednotlivé prvku jsou vybrané funkce.*

Obvykle jednotlivé geny genotypu odpovídají přímo komponentám či jejím funkčním částem ve fenotypu. Jelikož stochasticky generujeme spustitelné programy, je nutné velmi přesně popsat veškeré prvky, které budou při konstrukci a evaluaci chromozomu použity.

- `Terminály` - Jsou to prvky, které se ve stromové struktuře chromozomu vyskytují v listech. Patří mezi ně konstanty, proměnné a funkce bez argumentů s vedlejším účinkem. Proměnné zajišťují vstupy do vygenerovaného programu z vnějšího prostředí.
- `Funkce` - Množinu funkcí lze vybrat libovolně. Běžným postupem je kombinace aritmetických operátorů a aplikačně specifických funkcí, které pomáhají řešit zvolenou instanci problémů. U funkcí musíme zajistit konzistentní chování pro nestandardní vstupy tak, aby program mohl vždy proběhnout a funkce vrátila zpracovatelnou hod-

notu. Typickým příkladem je chování funkce pro dělení celých čísel v případě dělení nulou.

Při interpretaci chromozomů dochází k bijektivnímu zobrazení kódu v genech na popsané funkce a terminály.

2.3.2 Inicializace chromozomu

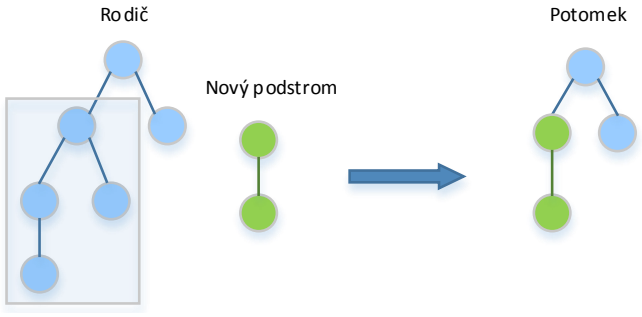
Každý výpočet evolučního algoritmu zahajujeme s počáteční populací. Taková populace obsahuje náhodně vygenerované jedince. Ti představují počáteční množinu řešení problému. U genetického programování je tedy nutné vytvořit iniciální posloupnosti instrukcí. Popišme základní metody inicializace s implicitním typem struktury strom. Uvažujme omezenou hloubku stromu h a množinu funkcí a terminálů.

- `Grow` - Vzniká nepravidelný strom. Každá větev je tvořena náhodně vybranými funkcemi a terminály. Její tvorba je ukončena buď terminálním prvkem při dosažení maximálního zanoření nebo náhodným výběrem terminálního prvku před dosažením maximálního zanoření.
- `Full` - Vzniká pravidelný strom. Každá větev dosahuje maximální hloubky a je tvořena náhodně vybranými funkcemi, zakončení je provedeno terminálním prvkem.
- `Ramped Half&Half` - Vytváříme nepravidelné stromy hloubky z množiny $\{2 \dots h\}$. Větve jednotlivých stromů jsou tvořeny metodami *Grow* a *Full* s pravděpodobností danou rovnoměrným lineárním rozložením.

2.3.3 Operátory

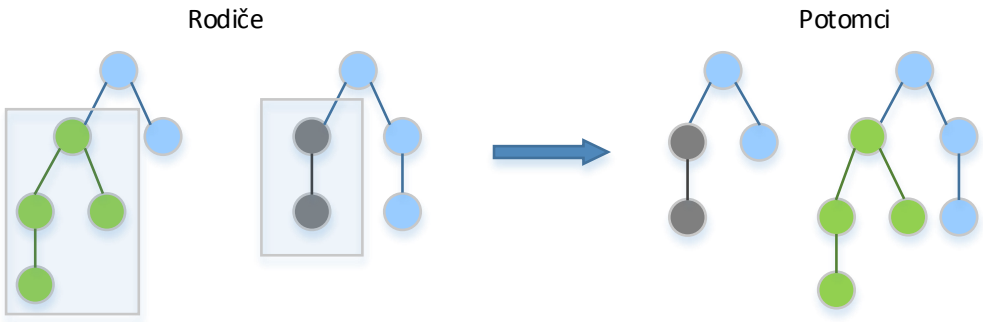
Mezi základní operátory patří křížení, mutace a selekce. Jejich chování je volně definovatelné, my zmíníme běžné příklady u stromové struktury.

Při *mutaci (obr. 2.2)* je nahrazen libovolný podstrom nově náhodně vygenerovaným podstromem. Tento jev diverguje populaci a na rozdíl od křížení nezachovává části podstromu, které by mohly nést těžiště řešení. Menší míru mutace představuje např. nahrazení jen jednoho náhodného uzlu grafu stochasticky vybraným uzlem stejného typu (toho se využívá u kartézského genetického programování *(kap. 2.4.2)*)



Obrázek 2.2: *Zleva můžeme vidět vybraný podstrom k nahrazení, náhodně vygenerovaný nahrazující podstrom a výsledného potomka.*

U *křížení* dochází k výměně dvou libovolných podstromů rodičů při vzniku dvou nových potomků. Na místa, odkud se odpojil vybraný podstrom se napojí podstrom druhého z rodičů *(obr. 2.3)*.



Obrázek 2.3: *Potomci vzniklí křížením rodičů pomocí vybraných podstromů.*

Selekční algoritmy mají za úkol vybrat silné jedince s přihlédnutím k pravděpodobnostnímu rozložení založeného na velikosti normalizované fitness. Konkrétní zvolené postupy budou rozebrány v popisu experimentálního nastavení pro vybraný problém *(kap. 6)*.

2.3.4 Ohodnocení fitness

Každou novou populaci je potřeba ohodnotit tak, abychom mohli korektně vybírat jedince pro proces křížení a mutací. Evaluace jedince odpovídá interpretaci a spuštění evolvovaného programu nad vstupními daty, obecněji nad modelem popisujícím data. Hrubá fitness f_h určuje absolutní hodnotu ohodnocení po ukončení evaluace jedince. Častěji se používá *normalizovaná fitness*.

Definice 2.11. *`Normalizovaná fitness`* f_n *- Mějme populaci jedinců* P*. Ti jsou ohodnoceni pomocí hrubé fitness* f_h*. Tu upravme do tvaru uspořádání se značením* f_s*, kde vyšší hodnota* f_s *představuje kvalitnějšího jedince. Tuto fitness normalizujme, potom* $\forall p \in P$ *:* $f_{n_p} = \frac{f_{s_p}}{\sum_{i=1}^{|P|} f_{s_i}}$*, kde* f_{n_p} *resp.* f_{s_p} *jsou normalizovaná fitness příslušející jedinci* p *resp. fitness ve tvaru uspořádání příslušející jedinci* p*.*

2.3.5 Nedostatky genetického programování

Genetické programování v sobě skýtá několik problematických prvků. Jedná se především o komplexnost vytvořeného programu, která je závislá přímo na limitované velikosti chromozomu. S potřebnou velikostí programu roste úměrně i velikost stavového prostoru, který je nutné prohledat k dosažení suboptimálního až optimálního řešení. Taková řešení mohou být takzvaně *přeučená*(v textu se objevuje i synonymní výraz přetrénovaná, více v def. 3.1). Vlivem přílišné specializace na trénovací množinu nejsou schopna zobecnění. V řešení se mohou postupem evoluce objevovat redundance neboli *introny*.

Definice 2.12. *`Intron`* *- Redundantní část kódu, která zpomaluje evaluaci a z hlediska vykonávaných instrukcí nemá žádný vliv na výsledek.*

Nelze vyloučit, že introny hrají roli při aplikaci operátorů mutací a křížení jako ochranný obal pro část kódu, která ve skutečnosti poskytuje hledané řešení. S introny souvisí i doba vyhodnocování programu. Pokud by byl program interpretován *Turingovým strojem*, nejsme schopni rozhodnout, zda stroj zastaví. Proto je ukončení interpretace ohraničeno shora časovým limitem, maximálním počtem provedených operací a podobně.

Druhým problémem obecně celé množiny evolučních algoritmů je crowding.

Definice 2.13. *`Crowding` - Jev, kdy se v populaci objeví jedinec s velmi vysokou hodnotou fitness. V případě, že je použit pouze triviální algoritmus výběru založený na proporcionálním zastoupení jedince v populaci na základě jeho fitness, v následných generacích se diverzita populace zmenší. Jejím těžištěm se stane jedinec s touto vysokou fitness a pokud nereprezentuje nejlepší řešení, může celý algoritmus konvergovat do lokálního minima.*

Řešením je komprimace či normalizace hodnot fitness, případně výběr jiné funkce pro výpočet pravděpodobnosti výběru jedince (linear ranking, turnajové metody).

2.3.6 Inovativní postupy

Uveďme několik nových postupů založených na genetickém programování.

- `Metagenetické programování` - Postup navržený *Jürgenem Schmidhuberem*. Evoluce postihuje nejenom chromozomy, ale i genetické operátory, které se jejím působením vyvíjejí. Specifikaci požadovaných vlastností výsledného programu ovlivní programátor pomocí definování vhodné fitness funkce. Můžeme proto vytvářet velmi obecné konstrukce a řešení škálovat na podproblémy.

- `Automaticky definované funkce (ADF) v genetickém programování` - Strom řešení obsahuje jako své podstromy podstromy funkcí a podstrom samotného programu, který funkce využívá. Programátor nadefinuje architekturu včetně počtu *ADF* a jejich parametrů. Další možnou úpravou je evolučně vygenerovat i architekturu, čímž se sníží počet nastavovaných atributů.

2.4 Kartézské genetické programování

Kartézské genetické programování je jednou z forem genetického programováním, sloužící pro automatizovaný návrh počítačových programů. Bylo vyvinuto v roce 1997 *Julianem Millerem*. Uceleně představeno bylo na konferenci *GECCO* 1999. Úpravy oproti genetickému programování se týkají především zpracování chromozomů a jejich reprezentaci [13].

2.4.1 Reprezentace chromozomu

Chromozom reprezentuje orientovaný acyklický graf. Při výpočtu uvažujeme několik omezujících podmínek na tvar grafu. Výsledná evolvovaná topologie bude mít n_r řádků a n_c sloupců. Celý systém má pevně daný počet n_i vstupů a n_o výstupů. Funkce se vybírají z množiny Γ, kde $|\Gamma| = n_f$ a mají až n_n parametrů (obvykle se uvažuje využití všech). Míru propojenosti celého systému určuje celočíselný parametr *L-back*.

Definice 2.14. *`L-back parametr` - Funkce (prvek topologie reprezentující funkci) v i. vrstvě (sloupci) může na své vstupy připojit výstup funkce ležící až v L předchozích sloupcích.*

Samotný chromozom je posloupnost genů. N-ticemi genů v celočíselném formátu $(I_{1k}, \ldots, I_{n_n k}, f_{cr})$, představujících k.uzel acyklického orientovaného grafu na r. řádku a c. sloupci chromozmu, reprezentujeme funkční komponenty ve fenotypu. Poslední výstupní n-tici genů zapisujeme ve formátu $(O_1, .., O_{n_o})$ *(obr. 2.4)*. f_{cr} je index ukazující do tabulky použitých funkcí. Vybraná funkce bude na c. řádku v r. sloupci topologie zpracovávat výstupy předchozích funkčních bloků, odkazovaných proměnnými $I_{1k}, .., I_{n_n k}$. Indexy, které

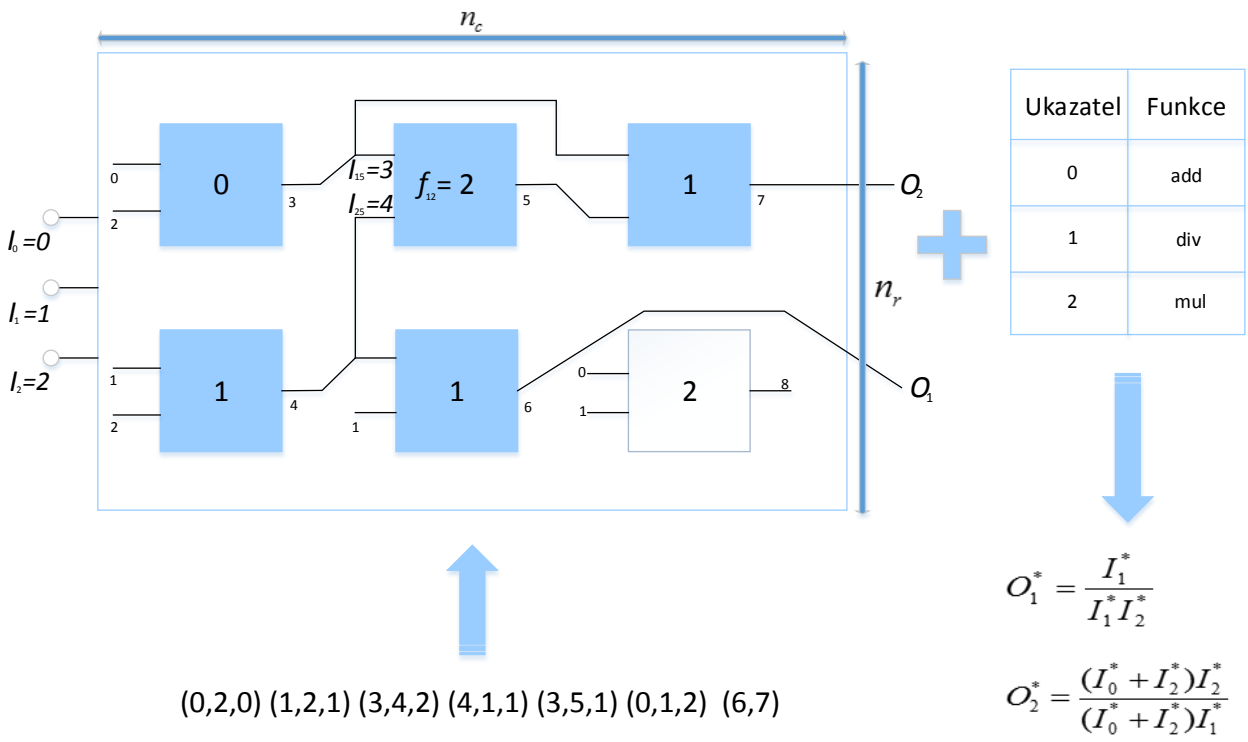
představují identifikátor výstupu funkce uzlu, jsou udělovány uzlům automaticky a systematicky, např. zleva doprava, shora dolů v obdélníkové struktuře. Výstup celého systému je dán ukazateli na výstupy vybraných funkcí (či na vstupy systému) v grafu. Ukazatele výstupu odpovídají proměnným $O_1, \ldots, O_{n_o}$. Takové zakódování nám dává limitu velikosti chromozomu $S_{lim} = n_r n_c (n_n + 1) + n_o$.

Pro naše potřeby zaveďme dereferencování proměnných ukazujících na výstupy jiných uzlů. Mějme $I_1 = 3$ odkazující na výstup určitého uzlu s indexem 3, dereferencovaná proměnná I_1^* udává aktuální velikost výstupu odkazovaného uzlu (se stejnou logikou můžeme použít ukazatel do tabulky funkcí).

Obrázek 2.4 znázorňuje dekódování chromozomu

$$(0,2,0)\ (1,2,1)\ (3,4,2)\ (4,1,1)\ (3,5,1)\ (0,1,2)\ (6,7)$$

do grafové struktury a její následnou interpretaci do podoby fenotypu. Bloky struktury jsou zakódované postupem shora dolu, zleva doprava. Vstupní body $0, 1, 2$ jsou pevně definované a nepodléhají mutacím. V chromozomu se na ně za pomoci genů můžeme odkazovat. V našem případě jsou parametry nastaveny následovně: $L = 3$, $n_r = 2$, $n_c = 3$, $n_i = 3$, $n_n = 2$ a $n_o = 2$. Každý z bloků ukazuje indexem do tabulky funkcí Γ, $\Gamma = \{add,\ div\ mul\}$. Fenotyp představují dvě výsledné funkce.



Obrázek 2.4: *Interpretace chromozomu. Detailně je popsán uzel na řádku* 1 *ve sloupci* 2. *Proměnná* I_{15} *odkazuje na index jeho prvního vstupu. Proměnné* I_0 *až* I_2 *reprezentují indexy vstupů chromozomu.*

Na obrázku 2.4 si všimněme v zakódování funkce s výstupem 8. Jedná se o *neaktivní uzel*, jelikož není použit ve fenotypu.

2.4.2 Operátory

Používá se pouze operátor *mutace*. Mutace může změnit typ funkce v uzlu, jakožto i způsob propojení uzlů. Takové změna můžou mít různý výsledek a podle toho mutace dále dělíme.

- `Neutrální mutace` - Fitness hodnota fenotypu se nezmění (např. mutace je provedena na neaktivním uzlu).
- `Adaptivní mutace` - Fitness hodnota nového fenotypu je odlišná od rodičovské.

Zobrazení chromozomu na výsledný fenotyp je surjektivní, tedy pro jeden fenotyp může existovat více rozdílných chromozomů. Mutace zastává funkci diverzifikace populace, zabraňující uvíznutí v lokálním minimu.

2.4.3 Obecný průběh algoritmu

U kartézského genetického programování obvykle pracujeme s populací jen několika jedinců. Pro demonstrační příklad použijme obecnou strategii $ES(\mu+\lambda)$. Tato strategie se v podobě $ES(5+5)$ a $ES(1+9)$ vyskytuje v experimentech v kapitole 6.

```
1 inicializuj čas t
2 definuj prázdnou množinu rodičů R
3 inicializuj novou populaci P(t) s počtem (μ +λ) jedinců
4 ohodnoť populaci P(t)
5 R = vyber μ nejlépe ohodnocených jedinců z P(t)

6 while není splněna definovaná ukončující podmínka algoritmu:
7    t' = t+1
8    inicializuj prázdnou populaci P(t')
9    for iterace in {0,...,λ}:
10       rodič = vyber náhodně rodiče z R
11       P(t') += mutuj rodiče
12    end for
13    t = t'
14    ohodnoť populaci P(t) a R

15    # vybíráme náhodně všechny jedince z P(t)
16    # ve snaze aktualizovat R
17    for jedinec J in P(t):
18       if R obsahuje generačně staršího a
                stejně ohodnoceného jedince jako J:
19          nahraď tohoto jedince z R za J
20       else if J je lépe ohodnocen než nejhůře ohodnocený jedinec z R:
21          nahraď nejhůře ohodnoceného jedince z R za J
22       end if
23    end for
24 end while
25 return nejlépe ohodnocený jedinec z R
```

Algoritmus 2: Obecný algoritmus kartézského genetického programování.

Upozorněme na důležitý fenomén z algoritmu 2. Vždy se snažíme mít množinu R rodičů obsazenou co nejvíce co nejnovějšími jedinci. Ve fázi ohodnocení populace se interpretuje struktura reprezentovaná pomocí chromozomů jedinců (obvykle nad řešeným modelem prostředí). Ukazatele na funkce jsou nahrazeny reálnými funkcemi, jsou dosazeny vstupní hodnoty (poskytuje model, jednoduchý model může být např. množina zadání a očekávaných výsledků) a provede se testovací výpočet.

Při použití evoluční strategie $ES(1 + \lambda)$ a jediné diverzifikační funkce v podobě mutace se řešení vyhýbají přirozeně crowdingu *(def. 2.13)* a selekční operátor může pracovat s hrubou fitness. To platí i pro dále popsané koevoluční techniky založené na kartézském genetickém programování.

Kapitola 3

Neuronové sítě

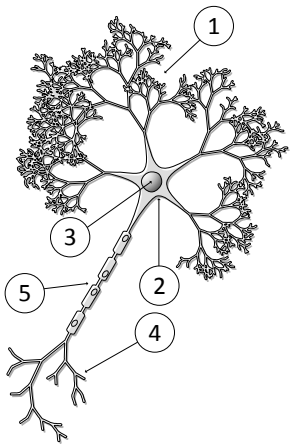
Metody umělých neuronových sítí [15] představují další z technik, která se nechala inspirovat přírodou. Svým principem se snaží napodobit chování mozku, přesněji se pak omezuje na neurony a struktury jimi vytvořené.

První zmínky o možnostech reprezentovat funkci neuronu pomocí matematického modelu se objevují ve 40.letech 20.století. První známý model navrhli *Warren McCulloch* a *Walter Pitts* v roce 1943. O 6 let později, roku 1949 *Donald Hebb* definuje pravidla učení takového neuronu. *Frank Rosenblatt* v roce 1958 představuje práci zabývající se *perceptrony*, které jsou využívány dodnes. Rozvoj teorie neuronových sítí je zastaven během 70.let, kdy *Marvin Minsky* publikuje práci o limitech takové teorie. Až na přelomu let 80. a 90., s rozvojem vícevrstevných sítí, jejich učení v podobě adaptace algoritmem *backpropagation*, metody neuronových sítí opět vychází do popředí. V poslední dekádě 20.století se objevují alternativní postupy reflektující nedostatky backpropagation, jako například deep learning pomocí omezeného *Boltzmannova stroje* či rekurentní nerunové sítě s *long-short term memory* rozvíjené *Jürgenem Schmidhuberem*. Na počátku 21. století se sítě využívají v rozmanitých oblastech od rozpoznávání, klasifikace, optimalizace, predikce až po dolování dat, jakožto v případech, kdy se obtížně definují matematické analytické modely, prostředí je dynamické, vyžadující paměť či souvislosti.

Umělé modely se tak stále více snaží přiblížit (a snad i překonat) svému předobrazu, což je bezesporu lidský mozek. Z toho vyplývají i bazální vlastnosti, které topologie neuronové sítě má.

- `Zpracování informace je distribuované a paralelní` - Informace jsou v síti rozmístěny mezi neurony, které paralelně data zpracovávají na základě zvolených pravidel.
- `Je vytvářena paměťová stopa` - Dlouhodobější znalosti a paměť v síti je udržována především pomocí síly vazeb mezi neurony.
- `Probíhá proces učení` - Při použití neuronových sítí nedefinujeme exaktní algoritmus transformace vstupní informace na výstupní. Místo toho dochází k procesu učení. Výsledná síť provede převod vstupních informací na výstup a nepřímo algoritmus odsimuluje, aniž bychom museli složitě definovat veškeré možné stavy, do kterých by se exaktní algoritmus mohl dostat. To mnohdy není ani možné.

3.1 Neuron



Obrázek 3.1: *Biologický neuron, který je předobrazem základních stavebních jednotek neuronových sítí. Na obrázku jsou postupně vyznačeny:* `dendrity (1)`, `soma (2)`, `jádro buňky (3)`, `axonální zakončení (4)` *a* `axon s mielinovou pochvou (5)`.

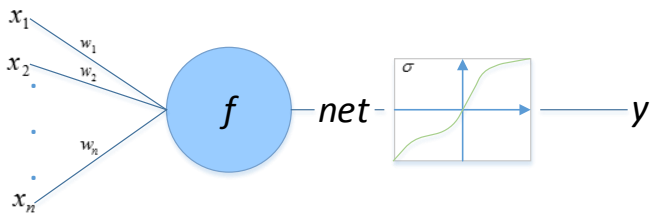
Neurony *(obr. 3.1)* jsou základní stavební jednotky mozku. Jejich hlavní funkcí je vytvářet mezi sebou spojení, *synapse*, a šířit elektricko-chemické vzruchy. V dnešní době považujeme za jednotku aktivity mozku počet spojení mezi neurony, nikoliv počet neuronů samotný.

Popišme si, jak vypadá šíření akčních potenciálů neuronovou sítí. Popis je zevrubný, ale přesně reflektuje potřeby transformace na matematický model. Neuron pomocí svých krátkých zakončení, *dendritů* vytváří synapse s ostatními neurony. Synapse je tvořena na jedné straně odstředivým *axonálním zakončením*, na druhé dostředivým výběžkem dendritu. Pokud se v těle neuronu (*somě*) vytvoří dostatečný potenciál, začne se šířit nervovým vláknem, neboli *axonem* s mielinovou pochvou, až k synapsi. V synaptické štěrbině tento elektrický vzruch vyvolá chemickou reakci. Uvolní se neurotransmitery, které jsou na druhé straně štěrbiny zachyceny specializovanými receptory. Aktivací postsynaptických receptorů se vytvoří postsynaptický potenciál v somě připojeného neuronu. Jestliže akumulace postsynaptických potenciálů přesáhne určitý práh, může se vyvolat další vzruch v daném neuronu. Ten se poté na krátký okamžik znecitliví.

Tento biologického procesu je popsán modelem neuronu *(kap. 3.2)*.

3.2 Umělý neuron

Obecný model neuronu vidíme na obrázku 3.2.



Obrázek 3.2: Matematický model neuronu

Popišme jeho základní části.

x_i ... `vstup neuronu`
Na vstupu může být výstup jiného neuronu nebo výstup z prostředí. $x \in \mathcal{R}$

w_i ... `váha spojení`
Váha $w_i \in \mathcal{R}$ udává vliv připojené části z hlediska aktivity neuronu. Často bývá prostřednictvím jedné z vah připojena konstantní hodnota - bias. Bias určuje práh aktivity neuronu.

f ... `bázová funkce neuronu`
Bázová funkce $f : \vec{w} \times \vec{x} \rightarrow net$ rozhoduje, jakým způsobím budeme určovat potenciál net neuronu.
Lineární bázová funkce $f = \sum_{i=1}^{n} w_i x_i$.
Radiální bázová funkce $f = \sqrt{\sum_{i=0}^{n} (x_i - w_i)^2}$

y ... `výstupní hodnota neuronu`

σ... `aktivační funkce`
Definujeme ji jako $\sigma : net \rightarrow y$. Určuje, jaký výstup bude v aktuálním čase neuron zprostředkovávat. Zohledňuje přitom hodnotu vah, vstupů a aktivační hodnotu neuronu. Ve všech zvolených ukázkách klademe pro bipolární hodnoty $a = -1$, $b = 1$, pro binární hodnoty $a = 0$ a $b = 1$. λ je empiricky zvolená konstanta.
Pro lineární bázovou funkci definujeme:

- `Skoková aktivační funkce`

$$y_{new} = \begin{cases} a & \text{pro net} < 0 \\ b & \text{pro net} > 0 \\ y_{old} & \text{pro net} = 0 \end{cases}$$

- `Spojitá sigmoidální aktivační funkce`

$$y = a + \frac{b - a}{1 + e^{-\lambda \text{net}}}$$

- `Spojitá aktivační funkce hyperbolického tangentu`

$$y = \frac{1}{2}\left(a + b + (b - a)tanh(\lambda \text{net})\right)$$

Pro radiální aktivační funkci definujeme aktivitu vzhledem k poloměru R.

- `Skoková aktivační funkce`

$$y_{new} = \begin{cases} 0 & \text{pro net} > R \\ 1 & \text{pro net} \leq R \end{cases}$$

- `Spojitá aktivační funkce`

$$y = e^{-\left(\frac{\text{net}}{\lambda}\right)^2}$$

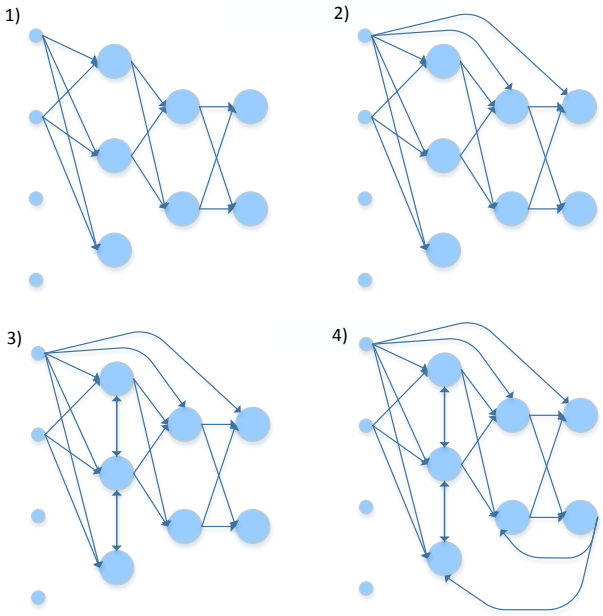
3.3 Topologie sítí a jejich charakteristika

Použitím jednoho neuronu s lineární bázovou funkcí lze v úloze zaměřené na klasifikaci rozdělit množinu vstupů na dvě podmnožiny, obecně přesněji rozdělit rovinu na dvě poloroviny hraniční hyperplochou.

Někdy ovšem prostor není lineárně separovatelný. Topologie sítí se proto zesložiťují přidáváním skrytých nelineárních vrstev, obsahujících nelineární aktivační funkce (existují samozřejmě alternativní řešení, jako hledání kernelu u *support vector machines* pro změnu dimenze prohledávaného prostoru atp.). Skryté vrstvy využívají nelineární aktivační funkce. V případě zachování lineárních aktivačních funkcí i u skrytých vrstev by byl výstup sítě opět jen lineární kombinací vstupních vektorů a klasifikační schopnosti sítě by se nezlepšily *(více v kap. 3.5)*.

Ne vždy provádíme klasifikaci. U asociativních přístupů *(více kap. 3.4)* může každá další vrstva zvyšovat komplexnost rozpoznání určitého charakteristického rysu.

Dalším důvodem pro použití složitější topologie je možná závislost posloupnosti vstupních dat zpracovaných sítí za delší časový úsek a výstupu sítě. Častým případem je predikce kurzů měn, vývoje akcií na burze atp.. Při řešení se používají rekurentní sítě. Rekurentní síť za své vstupy v jednotlivých vrstvách může uvažovat i předešlé výstupy. Hloubka rekurze nepřímo definuje množství předešlých informací, se kterými síť musí počítat, aby její vyhodnocování bylo správné.



Obrázek 3.3: *Topologie vícevrstvých sítí, v ukázkové topologii se nachází vstupní, skrytá a výstupní vrstva a propojení je provedeno jen u několika neuronů:* ***dopředná síť (1)***, ***acyklická síť (2)***, ***acyklická vrstvová síť (3)***, ***složitá rekurentní síť (4).***

Na obrázku 3.3 je několik základních způsobů propojení sítě. Systematicky se postupuje od *dopředné sítě*, kde jsou navzájem dopředně propojeny pouze neurony sousedních vrstev, přes *acyklickou síť* s dopředným propojením i přes více vrstev. U *acyklické vrstvové sítě* se přidávají propojení mezi neurony stejné vrstvy. *Rekurentní síť* přidává zpětnovazebné spojení a vyúsťuje v *plně propojenou síť*, kde jsou propojeny všechny neurony vzájemně.

Složitější topologie s sebou nesou značnou míru problémů spojených s učením *(více kap. 3.4 a 3.6.1)*, tak se samotným návrhem topologie. Počet a velikost vrstev sítě, vzájemné propojení neuronů a volba bázových a aktivačních funkcí se u konvenčních metod návrhu řeší empiricky, což nemusí vést vždy k nejlepším výsledkům. Oba zmíněné problémy řeší efektivně koevoluce kartézského genetického programování a neuronových sítí *(kap. 4)*.

3.4 Přístupy k učení a vyhodnocování sítí

Použití neuronových sítí v sobě zahrnuje dvě základní fáze. První fází je fáze učení. V té dochází k upravování synaptických vah sítě. Ve druhé fázi je určena odezva sítě. Ukažme si několik metod učení.

- `Korelační učení` - Učení typické například pro *Hopfieldovu sít či BAM* a další sítě reprezentující autoasociativní a heteroasociativní paměti. Trénovací množina je definována jako $T = \{\vec{i_1}, \vec{i_2}, .., \vec{i_p}\}$ respektive jako $T = \{(\vec{i_1}, \vec{d_1}), (\vec{i_2}, \vec{d_2}), .., (\vec{i_p}, \vec{d_p})\}$, kde $\vec{i}$ je

vstupní vektor a $\vec{d}$ požadovaný výstup. Při inicializaci dvouvrstvé *BAM* nastavujeme váhy pomocí trénovacích dvojic. Tím do sítě zaznamenáváme lokální energetická minima, ke kterým budeme chtít konvergovat i v případě částečně zašuměných hodnot. Definujeme také odpovídající aktivační hladinu bipolární aktivační funkce. Energetická funkce je definována jako $E = -\frac{1}{2}\sum_{i=1}^{i=n}\sum_{j=1}^{j=n} w_{ij}y_iy_j + \sum_{i=1}^{i=n}\theta_iy_i$, kde n je počet vstupů resp. výstupních neuronů sítě. θ je práh aktivity neuronu. Energetická funkce je v obměně stejná i pro Hopfieldovu síť. Lze ukázat, že je zdola omezená a že síť konverguje pro zvolený vstup a vybraný postup postupných úprav aktivit neuronů do svého energetického minima (na základě definice tzv. *Lyapunovy funkce*). Výsledek je asociovaný obraz pro použitý vzor. Platí přitom Hamingovo pravidlo výběru vzoru klasifikační funkcí ϕ. Mějme množinu binárních vzorů v o p prvcích a asociovaný binární obraz $\vec{i}$. Pak vybíráme $\vec{v_j} \in v$ s nejmenší Hamingovou vzdáleností d_h od $\vec{i}$, $\phi\left(\vec{i}\right) = \vec{v_j} \iff \forall k \in \{1..p\} : d_h\left(\vec{i}, \vec{v_j}\right) \leq d_h(\vec{i}, \vec{v_k})$.

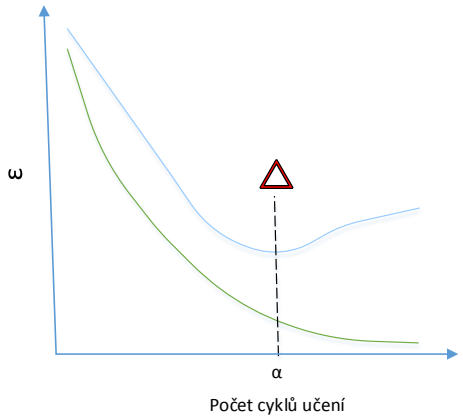
- `Soutěživé učení` - Na vstupy sítě jsou postupně přinášeny trénovací vzory. Síť má vnitřní vrstvu neuronů s radiálními bázovými funkcemi a spojitou aktivační funkcí. Je aktivován vždy pouze jeden neuron. Váhy si představme jako středy hyperploch. V každém korku učení se poté upravují váhy pouze vítězného neuronu tak, že střed tělesa se posune blíže zadanému vzoru.

- `Adaptační učení` - Mějme trénovací množinu $T = \{(\vec{i_1}, d_1), (\vec{i_2}, d_2), .., (\vec{i_p}, d_p)$. Odchylky výstupu sítě od požadovaného výstupu d definují chybovou funkci. Jednou z možností, jak upravit váhy sítě, je využití gradientu chybové funkce. Na základě gradientu definovaného parciálními derivacemi chybové funkce podle vah upravujeme postupnými úpravami váhy všech vrstev sítě např. metodou backpropagation, podrobněji popsanou v kapitole 3.6.1.

Všimněme si, že v některých případech, například u adaptačního učení, trénovací množina obsahuje i očekávaný výstup sítě. Potom je cílem sítě *generalizovat*, aby klasifikovala správně i vozrky, které se přímo nevyskytly v trénovací množině. Takové učení nazýváme *učení s učitelem*. Naopak například u soutěživého učení se síť snaží vzorky *kategorizovat* do množin podle jejich vzájemné podobnosti, nemá přitom dopředu informaci, jaké rozdělení je očekávané. Jedná se o učení *bez učitele*.

Při učení sítě může dojít k přeučení.

Definice 3.1. ***`Přeučení`*** *- Přeučení je jev, který může nastat během učení sítě. Chyba sítě se při vyhodnocení na trénovací množině dat během procesu učení stále snižuje. To platí i o chybě na množině testovacích dat. Jakmile se chyba na množině testovacích dat začne zvětšovat, zatímco chyba na množině trénovací stále klesá, jedná se o přeučení. Tento jev svědčí o nedostatečné schopnosti generalizace sítě. Může být způsoben více faktory, např. nesprávně zvolenou trénovací množinou dat či špatným návrhem topologie sítě. Vliv může mít i počáteční inicializace vah a počet kroků učení. Je nutné detekovat tzv. bod časného zastavení, kdy míra chybovosti sítě na trénovací množině stále klesá, ale schopnost zobecnit se zhoršuje (obr. 3.4).*



Obrázek 3.4: *Graf závislosti počtu cyklů učení sítě na vývoji chyby klasifikace ω. Modrá křivka odpovídá vývoji učení na testovací množině, zelená křivka odpovídá učení na množině trénovací. V bodě α došlo k přetrénování sítě.*

Dalšími problematickou pasáží např. adaptačního učení je uvíznutí v lokálním minimu chybové funkce. Představme si n rozměrnou topologii definující závislost velikosti chyby vyhodnocení vzorku na nastavení vah sítě. Při příliš rychlé konvergenci vah při procesu učení může síť v jednom z minim uvíznout. Lze použít postupů simulovaného žíhání nebo využít momentum pro snížení pravděpodobnosti takové události. Mnou vyzkoušená a velmi efektivní metoda je také dynamické měnění kroku učení, kdy při změně směru gradientu chyby vyhodnocování pro aktuální nastavení hodnot vah sítě zmenšujeme krok (a naopak při konstantním směru můžeme krok zvětšit).

Vyhodnocení je stejně jako učení specifické pro danou síť. Postup tkví v postupném vyhodnocování aktivit neuronů, míra paralelizmu je dána topologií sítě. Výsledek je reprezentován aktivitou výstupních neuronů. Podrobněji popíšeme tento proces u vícevrstvých sítí v kapitolách 3.6.1 a 3.6.2.

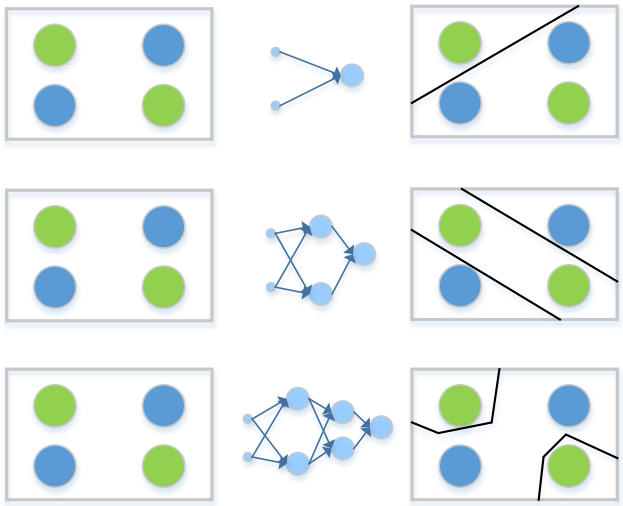
3.5 Klasifikační možnosti sítí

Ukázali jsem si rozdílné topologie sítí, způsob aktivace neuronů a také přístupy k učení. V rámci adaptačních sítí tyto faktory přímo ovlivňují možnosti klasifikace sítě. Jiným pohledem na neuronovou síť je uvažování celé sítě jako matematické funkce. Přesněji pak polynomu, ve kterém upravujeme velikosti hodnot koeficientů proměnných.

Na obrázku 3.5 jsou zástupci dopředných sítí reprezentující adaptační učení. Pokud klasifikujeme pomocí jednoho neuronu se dvěma vstupy s lineární bází, polynom reprezentuje přímku. Přidáváním vrstev s nelineárními aktivačními funkcemi a úpravou počtu propojení sítě můžeme zvyšovat stupeň polynomu odpovídajícího neuronové síti. Vzniká topologie, kterou separujeme prostor na podprostory. Dvě skryté vrstvy jsou schopny klasifikovat množinu definovanou konvexním polygonem, tři a více vrstev klasifikují libovolnou množinu. Toto tvrzení souvisí s *Kolmogorovovým teorémem.*

Teorém 3.2. *`Kolmogorovův teorém` - Teorém říká, že libovolnou funkci n proměnných*

lze vyjádřit pomocí lineárních kombinací a jedné opakovaně použité nelineární monotonně rostoucí funkce jedné proměnné.



Obrázek 3.5: *Ukázky možnosti separování prvků problému XOR, který není lineárně separovatelný. Demonstrace možných výstupů dělící křivky při aplikaci vícevrstevných sítí s nelineární aktivací. Cílem je oddělit do skupin stejně zbarvené prvky.*

V případě rekurentního zapojení uvažujeme i předešlé vstupy sítě, tudíž můžeme např. řešit i diferenciální rovnice a aproximovat budoucí vývoj posloupností.

3.6 Vícevrstevné sítě

Použití vícevrstvých sítí s nelineárními funkcemi ve skrytých vrstvách si vyžádalo i navržení technik pro jejich učení.

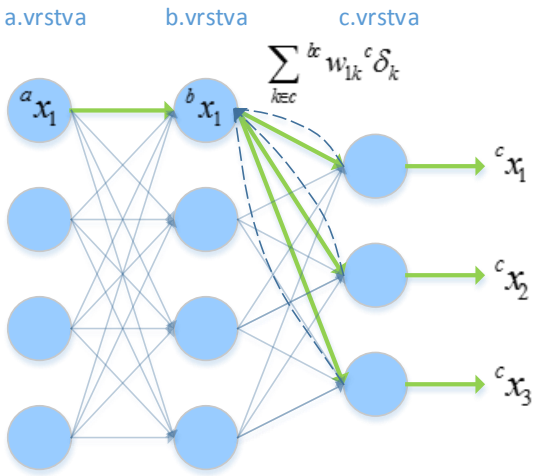
Předvedeme si dvě techniky. První z nich je *backpropagation (kap. 3.6.1, [15])*, typickou pro adaptační učení. Backpropagation poukázala i na řadu problémů související s limitem stávajících výpočetních zdrojů a rychlostí šíření chyby výpočtu. Je tudíž dobrou ukázkou, proč se začaly objevovat i další alternativy učení složitějších topologií. Touto alternativou budou techniky koevoluce *CGP* a neuronových sítí.

Druhou metodou představenou v této kapitole je využití *omezeného Boltzmannova stroje* ve více vrstvách pro asociativní učení. Technika podobná metodě postupného učení jednotlivých vrstev bude použita u nepřímé zakódování *Developmental Network* v kapitole 4.1. Zde se síť bude postupně rozvíjet z malé počáteční konfigurace.

3.6.1 Učení s backpropagation

Pro ujasnění terminologie definujme několik základních proměnných. ${}^{i}x_{j}$ je aktivita neuronu v i. vrstvě na j pozici. Stejnou indexaci budeme přiřazovat i potenciálu net. Váhu mezi i. neuronem vrstvy a a j. neuronem vrstvy b u spojení vycházejícího z a do b popišme jako ${}^{ab}w_{ij}$. Ukažme úpravu velikosti vah v případě, že trénovací vektor je definován jako

$T = \{(\vec{i_1}, \vec{d_1}), (\vec{i_2}, \vec{d_2}), .., (\vec{i_p}, \vec{d_p})\}$. Pro jednoduchost uvažujme binární formát vstupu sítě a lineární aktivační funkci výstupní vrstvy sítě (obor hodnot $H_\sigma \in < 0; 1 >$). Vzorový úsek sítě bude obsahovat 3 vrstvy a, b a c *(viz. obr. 3.6)*.



Obrázek 3.6: *Ukázka šíření chyby vnitřními vrstvami sítě.*

Nejprve síť postupně po vrstvách zpracovává vstupní vektor. Přitom se v každé vrstvě postupně mění aktivita neuronů na základě nelineární aktivační funkce $\sigma(net) = \frac{1}{1+e^{-net}}$, kde potenciál neuronu je ${}^{b}net_j = \sum_{i \in a} {}^{ab}w_{ij}\,{}^{a}x_i$. Ve chvíli, kdy se dostaneme k výstupní vrstvě, porovnáme hodnoty aktivit daných lineární aktivační funkcí s $\vec{d}$. Ve skrytých i výstupních vrstvách mohou být samozřejmě rozličné typy funkcí, my uvažujeme konkrétní příklad, přesto uvedeme základní rovnice pro učení i v obecném znění.

Pomocí $\vec{d}$ zadefinujeme chybu vyhodnocení. Našim záměrem bude změnit hodnotu vah tak, aby se chyba minimalizovala. Celkovou chybu sítě určíme pro vybraný vzorek obecně jako $E = \sum_{k \in \text{výstupní vrstva}} \frac{1}{2}(d_k - {}^{\text{výstupní vrstva}}x_k)^2$. Chybu budeme distribuovat sítí a upravovat váhy spojení. Znaménko úpravy určíme pomocí derivace energetické funkce podle váhy spojení. Cílem je dostat se do minima chybové funkce E. Proces načtení dat, určení chyby a její distribuce se obvykle provádí opakovaně, nežli chyba neklesne pod stanovenou mez. Techniky pro optimalizaci úpravy vah při distribuci chyby pomocí momenta atp. nebudeme diskutovat, rozeberme ale způsob šíření hodnoty úpravy vah definovanou topologií.

Rozlišme případ, kdy určujeme derivaci chyby výstupních vrstev (v našem příkladu s lineární aktivační funkcí) a kdy skrytých vrstev (v našem příkladu s nelineární aktivační funkcí) . Obecně platí

$$\frac{\partial E}{\partial {}^{ab}w_{ij}} = \frac{\partial E}{\partial {}^{b}x_j} \frac{\partial {}^{b}x_j}{\partial {}^{b}\text{net}_j} \frac{\partial {}^{b}\text{net}_j}{\partial {}^{ab}w_{ij}}$$

Dále zaveďme pomocnou proměnnou δ. Pro proměnnou δ bude platit, že

$${}^{b}\delta_j = \frac{\partial E}{\partial {}^{b}\text{net}_j} = \frac{\partial E}{\partial {}^{b}x_j} \frac{\partial {}^{b}x_j}{\partial {}^{b}\text{net}_j} \tag{3.1}$$

pak

$$\frac{\partial E}{\partial {}^{ab}w_{ij}} = {}^{a}x_i\,{}^{b}\delta_j \tag{3.2}$$

V případě lineární výstupní vrstvy sítě mějme (použijeme metody matematické analýzy [14])

$$
\begin{aligned}
{}^{b}\delta_j &= ({}^{b}x_j - {}^{b}d_j)\frac{\partial^b x_j}{\partial^b \mathrm{net}_j} && (3.3)\\
\frac{\partial E}{\partial^{ab} w_{ij}} &= -({}^{b}d_j - {}^{b}x_j)^{a}x_i \frac{\partial^b x_j}{\partial^b \mathrm{net}_j} && (3.4)
\end{aligned}
$$

Byla provedena substituce δ do rovnice 3.2. Jelikož provádíme derivaci lineární aktivační funkce, výsledek derivace $\frac{\partial^b x_j}{\partial^b \mathrm{net}_j}$ odpovídá konstantě, kterou zahrneme do velikosti kroku, jímž se budeme posunovat po funkci E k minimu. Pokud je derivace E kladná, jedná se o místo růstu funkce, proto budeme od hodnoty stávající váhy výsledek derivace násobené infinitezimálním krokem ϵ odčítat, pokud je parciální derivace záporná, budeme krok přičítat, pro náš konkrétní příklad a výstupní vrstvu c mějme

$$
{}^{bc}w_{ij} = {}^{bc}w_{ij} - \epsilon\frac{\partial E}{\partial^{bc} w_{ij}} = {}^{bc}w_{ij} + \epsilon\ ({}^{c}d_j - {}^{c}x_j)\ {}^{b}x_i \frac{\partial^c x_j}{\partial^c \mathrm{net}_j} \qquad (3.5)
$$

Pro skryté vrstvy (v našem příkladu s nelineární aktivační funkcí) bude výpočet mírně složitější. Předešleme využití myšlenky, že gradient sumy chybové funkce odpovídá sumě gradientů chyby z prvků vyšší vrstvy. Rozeberme postupně a obecně parciální derivace pomocné proměnné δ. Nejprve vyjádřeme část $\frac{\partial E}{\partial^b x_j}$ z rovnice 3.1 pomocí sumy gradientů předchozích vrstev.

$$
\begin{aligned}
\frac{\partial E}{\partial^b x_j} &= \sum_{k \in \text{c.vrstva}} \frac{\partial E}{\partial^c \mathrm{net}_k}\frac{\partial^c \mathrm{net}_k}{\partial^b x_j}\\
\frac{\partial E}{\partial^b x_j} &= \sum_{k \in \text{c.vrstva}} \frac{\partial E}{\partial^c \mathrm{net}_k}\,{}^{bc}w_{jk}\\
\frac{\partial E}{\partial^b x_j} &= \sum_{k \in \text{c.vrstva}} {}^{c}\delta_k\,{}^{bc}w_{jk}
\end{aligned}
$$

Pro výpočet δ, potažmo $\frac{\partial E}{\partial^{ab} w_{ij}}$, skryté vrstvy proto obecně platí

$$
\begin{aligned}
{}^{b}\delta_j &= \frac{\partial^b x_j}{\partial^b \mathrm{net}_j} \sum_{k \in \text{c.vrstva}} {}^{c}\delta_k\,{}^{bc}w_{jk} && (3.6)\\
\frac{\partial E}{\partial^{ab} w_{ij}} &= {}^{a}x_i \frac{\partial^b x_j}{\partial^b \mathrm{net}_j} \sum_{k \in \text{c.vrstva}} {}^{c}\delta_k\,{}^{bc}w_{jk} && (3.7)
\end{aligned}
$$

Jestliže použijeme aktivační nelineární funkci $\sigma(net) = \frac{1}{1+e^{-net}}$, potom

$$
\frac{\partial^b x_j}{\partial^b \mathrm{net}_j} = {}^{b}x_j(1 - {}^{b}x_j)
$$

Pak parciální derivace energetické funkce podle vah ve skrytých vrstvách odpovídá hodnotě

$$
\begin{aligned}
{}^{b}\delta_j &= {}^{b}x_j(1 - {}^{b}x_j) \sum_{k \in \text{c.vrstva}} {}^{c}\delta_k\,{}^{bc}w_{jk} && (3.8)\\
\frac{\partial E}{\partial^{ab} w_{ij}} &= {}^{a}x_i\,{}^{b}x_j(1 - {}^{b}x_j) \sum_{k \in \text{c vrstva}} {}^{c}\delta_k\,{}^{bc}w_{jk} && (3.9)
\end{aligned}
$$

Úprava vah je definována jako

$$ {}^{ab}w_{ij} \quad = \quad {}^{ab}w_{ij} - \epsilon \, {}^{a}x_i \, {}^{b}x_j (1 - {}^{b}x_j) \sum_{k \in \text{c vrstva}} {}^{c}\delta_k \, {}^{bc}w_{jk} \tag{3.10} $$

Šíření chyby vnitřními vrstvami je zobrazeno na obrázku 3.6. Podotkněme, že jsme ukázali jak obecné rovnice upravení vah při učení ve skrytých *(rovnice 3.6 a 3.7)* i výstupních vrstvách *(rovnice 3.1 a 3.2)*, tak konkrétní rovnice pro výstupní vrstvu s lineární aktivační funkcí *(rovnice 3.3, 3.4 a 3.5)* a skrytou vrstvu se sigmoidální aktivační funkcí *(rovnice 3.8, 3.9 a 3.10)*.

Problematická pasáž použití *backpropagation* spočívá v sumě gradientů pro hlouběji zanořené vrstvy. Uvažme, jakou hodnotu bude mít δ ve 3. vrstvě sítě s nelineárními vrstvami v našem konkrétním příkladu

$$ {}^{a}\delta_i = {}^{a}x_i (1 - {}^{a}x_i) \sum_{j \in b} \sum_{k \in c} {}^{ab}w_{ij} \, {}^{b}x_j (1 - {}^{b}x_j) \, {}^{bc}w_{jk} ({}^{c}x_k - {}^{c}d_k) \frac{\partial^c x_k}{\partial^c \text{net}_k} \tag{3.11} $$

jestliže hodnoty proměnných x a d jsou menší nebo rovny 1. Celá síť bude konvergovat k optimu velmi pomalu.

Představme si rekurentní neuronovou síť. Potřebná délka sledované sekvence vzorků v čase nepřímo určuje počet vrstev sítě, například při rozbalování sítě metodou *backpropagation through time*. Opět budeme muset řešit problém s pomalou konvergencí vah díky evaporaci hodnoty gradientu (stejně jako v rovnici 3.11) .

3.6.2 Učení hlubokých neuronových sítí s omezeným Boltzmannovým strojem

V poslední kapitole k tématu neuronových sítí si ukažme vícevrstvý model neuronové sítě založené na *omezeném Boltzmannově stroji (*RBM) [12]. Tato síť patří mezi sítě s korelačním typem učení. Zde se proto neuplatní přímo metoda backpropagation popisovaná v kapitole 3.6.1. Síť, jak uvidíme, se bude učit postupně po vrstvách. Tento fakt zohledníme při tvorbě sítě pomocí *Developmental Network* v kapitole 4.

Boltzmannův stroj pracuje s pojmy pravděpodobnosti a podmíněné pravděpodobnosti vrstvy sítě. Až zavedením omezeného stroje, který nedefinoval žádná spojení v rámci skryté vrstvy dvouvrstvé topologie, se učení této topologie stalo dostatečně rychlým a vhodným pro další experimenty.

Definujme viditelnou vrstvu sítě $\vec{v}$ a skrytou vrstvu $\vec{h}$, vzájemně plně propojené. Našim cílem je pro trénovací vektory nalézt nastavení vah a případně biasů tak, aby jejich pravděpodobnost byla maximální (a zároveň celková energie sítě minimální), jak vyjadřuje rovnice 3.12. Definujme hodnoty aktivity i.neuronu ve viditelné vrstvě jako v_i a hodnoty aktivity j.neuronu ve skryté vrstvě jako h_j. Váhu spojení mezi j. neuronem skryté vrstvy a i.neuronem viditelné vrstvy definujme jako w_{ij}. Pravděpodobnost vrstvy $\vec{v}$ při daném trénovacím vzoru určíme pomocí energie sítě. Iterujeme přes všechna možná spojení skryté a viditelné vrstvy RBM, odtud $E(\vec{v}, \vec{h}) = -\sum_{i,j} h_i w_{ij} v_j$. Pravděpodobnost rekonstrukce odpovídá

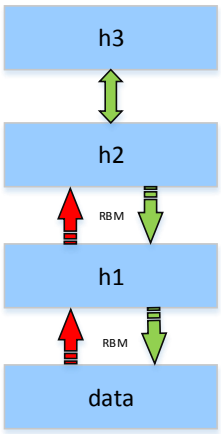
$$ p(\overrightarrow{v}) = \frac{\sum_{\overrightarrow{g}} e^{-E(\overrightarrow{v}, \overrightarrow{g})}}{\sum_{\overrightarrow{u}, \overrightarrow{g}} e^{-E(\overrightarrow{u}, \overrightarrow{g})}} \tag{3.12} $$

Sumy v čitateli a jmenovateli zahrnují veškeré možné konfigurace nastavení vektorů (pro jednoduchost uvažujme inicializaci binárními hodnotami). Chceme maximalizovat pravděpodobnost p, pro úpravu vah proto definujme (zlogaritmování hodnoty p představuje výhodu nejenom pro výpočetní náročnost operací, ale je i klíčem pro vysvětlení učení pomocí postupných rekonstrukcí vstupního trénovacího vzoru, pro větší detaily čtenáře odkažme na [11])

$$\frac{\partial log(p(\overrightarrow{v}))}{\partial w_{ij}} =< v_i h_j >^0 - < v_i h_j >^\infty \tag{3.13}$$

Učení probíhá tak, že se postupně určuje aktivita skryté a následně z ní viditelné vrstvy *(rovnice 3.13)*, podle kterých se upravují parametry sítě. Síť takto konverguje do energetického minima. Pokud necháme *RBM* zpracovávat střídavě vstupní a skrytou vrstvu dostatečně dlouho, dostaneme hodnoty, které určí směr gradientu pravděpodobnostní funkce a tak i způsob úpravy vah. Indexy 0 a ∞ v rovnici 3.13 určují, v kolikátém kroku rekonstrukce (krok zahrnuje přenos dat do skryté vrstvy a zpět) se daný součin provede. Uvědomme si, že rekonstruované data již představují model reálných dat z iterace 0.

Rádi bychom z časového hlediska aktualizaci vah prováděli po jednom kroku rekonstrukce. Ukázalo se, že ačkoliv aktualizace provedená pomocí $\Delta w_{ij} = \epsilon(< v_i h_j >^0 - < v_i h_j >^1)$ nemá stejný gradient, který je dán v rovnici 3.13, přesto pro metodu učení pracuje velmi dobře. Dokážeme tak rychle učit jednotlivé vrstvy hluboké sítě. Tyto vrstvy jsou totiž složeny právě pomocí *RBM*. Skrytá vrstva $\overrightarrow{h}$ *RBM* je definována tak, aby byla schopna identifikovat charakteristické rysy vrstvy vstupní. Zároveň máme-li natrénovanou *RBM* s trénovacími vektory, pro které je vysoké $p(\overrightarrow{v})$, bude vysoká míra pravděpodobnosti přisouzena i vektoru tvořeném skrytou vrstvou, od které, jak bylo řečeno, vyžadujeme extrakci obecných rysů vstupu. Pokud výstup jedné vrstvy použijeme jako vstup pro další vrstvu, na kterou opět aplikujeme metodiku učení (vrstvy musí mít správnou velikost, aby docházelo k extrakci rysů), dostaneme vícevrstvou síť schopnou lépe škálovat rysy (viz obr. 3.7).



Obrázek 3.7: *Fragment zachycující několik vrstev sítě. Při rekonstrukci probíhá vzorkování výstupu sítě v nejvyšší vrstvě. Červeně značené spojení slouží pouze pro učení sítě.*

Dodejme, že obvyklým způsobem (v roce 2014) se používají *RBM* v rámci hlubokého učení ve spojení s konvolučními neuronovými sítěmi. Detekované rysy se mohou v poslední vrstvě takové sítě použít při učení s učitelem pro správnou klasifikaci. Typickým příkladem je projekt *cuda-convent* (`https://code.google.com/p/cuda-convnet/`), umožňující rozpoznávání v obraze.

Kapitola 4

Koevoluce genetického programování a neuronových sítí

Koevoluce představuje synergii dvou předešlých technik, jmenovitě neuronových sítí a genetických algoritmů.

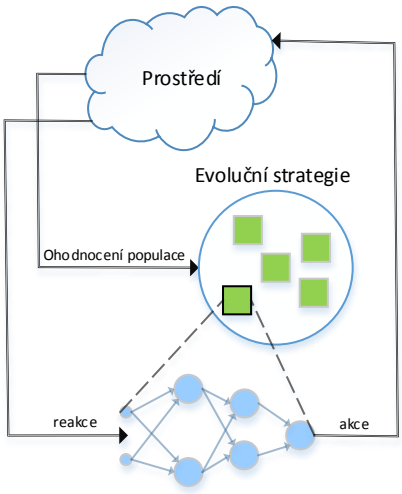
Jak jsme mohli vidět v kapitole 3.5, v dnešní době existuje velké množství specializovaných modelů pro řešení specifických úloh. Zároveň jsme v kapitole 3.6.1 nastínili problémy učení větších topologií na základě gradientní metody pomocí adaptačního učení.

Aplikační požadavky na ideální neuronovou síť jsou ovšem odlišné. Cílem je, aby síť byla univerzálním nástrojem schopným řešit zadané problémy. Aplikačně specifické parametry by měly tvorbu a učení sítě ovlivňovat co nejméně v zájmu zachování obecnosti s maximální potlačením NFL teorému *(2.1)*. Při učení nechceme používat metodu s učitelem, kdy u trénovacích dat uvádíme očekávané výsledky, jelikož u složitých nelineárních problémů uvažujeme rozsáhlý stavový prostor, těžko pokrytelný takovou množinou. Očekáváme, že síť bude interagovat s prostředím a podle reakce na něj upraví své chování.

Myšlenkou koevoluce *CGP* s neuronovými sítěmi je evolvovat zakódování popisující síť tak, aby výsledná síť pracovala co nejlépe. My popíšeme dva základní typy a to přímé zakódování sítě [7] *(kap. 4.2)* a nepřímé zakování sítě pomocí *Development Network* [6] *(kap. 4.1)*. Ačkoliv obě techniky kódování popisují výslednou síť odlišně, vždy ovlivňují tvar její topologie a schopnost učení. Pro reprezentaci zakódování jsme zvolili *CGP* chromozomy.

Definujme základní myšlenku koevoluce tak, jak ji znázorňuje obrázek 4.1. Pomocí chromozomů zakódujeme vlastnosti neuronové sítě. Tyto chromozomy můžeme transformovat v reálnou neuronovou síť představující fenotyp. S touto sítí vyhodnocujeme reakce prostředí popsané modelem M a aplikujeme na ně akce definované výstupy sítě. Dle stavu prostředí po aplikaci akcí sítě vyhodnotíme kvalitu daného chromozomu. S fitness pracuje evoluční část koevoluce a evolvuje za pomoci mutací v každé generaci co nejlépe nová zakódování sítě (jedince), přitom reflektuje popis v algoritmu 2.

Techniky koevoluce se tak s úspěchem uplatňují při tvorbě neuronových sítí určených k rozeznávání vzorů, řízení chemických procesů atp..



Obrázek 4.1: *Obecný průběh koevoluce. Ukázka transformace vybraného chromozomu (zelený čtverec) populace na neuronovou síť a její interakce s prostředím. Na populaci je přitom aplikována evoluční strategie, která reflektuje ohodnocení jedinců.*

Pro přehlednost je v příloze A uvedena tabulka nejčastěji použitých symbolů.

4.1 Nepřímé zakódování sítě pomocí `Developmental Network`

Klasický pohled na neuronovou síť v sobě zahrnuje pevně danou strukturu neuronů a jejich propojení, která je schopna učení. Pomocí koevoluce s nepřímým zakódováním se pokusíme více přiblížit skutečné biologické podstatě učení. Chtěli bychom jedince, jejichž genetický materiál bude kódovat vlastnosti jednotlivých komponent neuronů (axon, soma atd.), ne přímo celé sítě. Takový genetický materiál se bude upravovat během generací vývoje celé populace, každý jedinec bude mít svůj unikátní. Na základě tohoto genetického materiálu a interakce s prostředím se během života jedince bude rozvíjet jeho neuronová síť. Bude se učit, učením se budou posilovat vybraná neuronová spojení. Nepoužívané neurony zaniknou, nebo se počty jejich dendrálních výběžků zmenší, naopak více používané neurony počet svých axonálních zakončení a dendrálních výběžků budou zvětšovat.

Mluvíme o velmi komplexním modelu s velmi propracovaný chováním. Definujme několik základních vlastností neuronů. Tyto vlastnosti budou kódovány chromozomy a budou unikátní pro každého jedince. Celkem mějme 7 chromozomů, každou vlastnost reprezentuje právě jeden (všimněme si, že v předchozích popisech byl jedinec reprezentován jedním chromozomem, nyní je jich 7 pro jednoho jedince). Ukažme základní problémy, které řídí chromozomy

1. Interakce dendrálních výběžků jednoho dendritu s okolím.
2. Zpracování signálu v jádře neuronu a rozhodnutí o další distribuci signálu.
3. Funkce synaptického spojení.
4. Růst dendritů, změna jejich počtu a polohy jejich výběžků.
5. Růst a změna počtu axonálních zakončení.

6. Určení váhy spojení mezi axonálními zakončeními a dendrálními výběžky.

7. Vytvoření a odstranění neuronů.

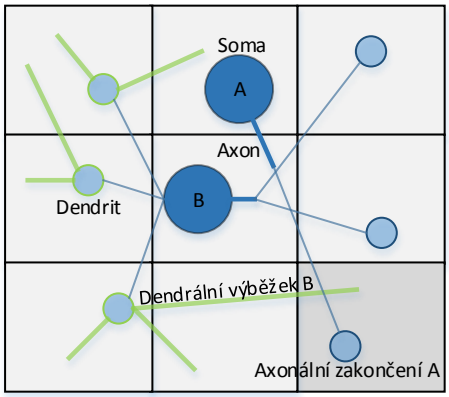
Každý z chromozomů pracuje se specifickou množinou komponent neuronu a je stejný pro všechny neurony jednoho jedince. Tyto chromozomy podléhají evolučnímu procesu. Každý chromozom je v podstatě jeden program vytvořený kartézským genetickým programováním (*CGP*). Uvědomme si, že chromozomy neříkají přímo nic o tom, jak se síť má učit. O tom rozhodne interakce s prostředím (modelem) během života jedince. Chromozomy definují chování neuronů, způsob distribuce signálu neuronovou sítí a změnu tvaru její topologie. Kromě chromozomů a neuronů použijeme ještě jeden významný prvek, *mřížku*. Jedná se o plochu, na které jsou neurony rozmístěny a která určuje, k jakým spojením mezi nimi může dojít.

Postupně jednotlivé aspekty řešení formálně popíšeme v následujících kapitolách. Budeme se zabývat rozborem struktury neuronu *(kap. 4.1.1)*, detailně popíšeme funkce mřížky *(kap. 4.1.1)* a jednotlivých skupin chromozomů *(kap. 4.1.3, 4.1.4, 4.1.6 a 4.1.5)*. Na závěr této kapitoly rozebereme celý algoritmus tak, jak ho lze obecně aplikovat *(kap. 4.1.9)*. V textu budeme také uvádět konkrétní nastavení, se kterými jsme dále experimentovali. Je to proto, aby si čtenář dokázal vytvořit ucelenou představu o algoritmu.

4.1.1 Komponenty neuronu a *mřížka*

Na začátku celého výpočtu inicializujeme neurony do *mřížky*.

Definice 4.1. ***Mřížka*** *- Jedná se o mřížku ve tvaru obdélníku. Spojení mezi neurony A a B je možné jen v případě, že dendrální výběžky neuronu A a axonální zakončení neuronu B sdílí stejnou buňku mřížky. V takovém případě může dojít k výměně potenciálu mezi neurony A a B. Definujeme dvě proměnné mřížky G a to počet jejich sloupců G_c a řádků G_l (obvyklé nastavení je $G_c = 3$, $G_l = 4$). Příklad spojení neuronů můžeme vidět na obrázku 4.2.*



Obrázek 4.2: *Příklad spojení neuronu A a B. V tomto případě dojde ke spojení ve zvýrazněném pravém dolním poli mřížky, do kterého svým axonálním zakončením zasahuje neuron A a dendrálním výběžkem neuron B.*

Inicializace neuronů probíhá náhodně. Neuron se skládá z několika komponent. Jsou jimi (v závorce uvádíme index, který bude označovat atributy a konstanty vztahující se

k dané komponentě (viz. kapitola 4.1.2)): soma (s), axon (a), axonální zakončení (az), dendrit (d) a dendrální výběžky (dv). Jak bylo řečeno, při inicializaci neuronu se obvykle používají stochastické metody. Hodnoty maximálního počtu jednotlivých komponent pro jeden neuron jsou označeny těmito konstantami (iniciační počty komponent jsou voleny náhodně, nesmí přesáhnout uvedenou maximální hranici, námi použitá nastavení uvádíme v závorce): maximální počet axonálních zakončení ($N_{az} = 5$), počet dendritů ($N_d = 5$) a výběžků pro jeden dendrit ($N_{dv} = 3$). Komponenty se do mřížky rozmístí a tvoří iniciální neuronovou síť jedince až o N_n neuronech ($N_n = 5$). Jedná se o analogii s mozkem mláděte. Jeho neuronová síť není nikterak vyvinutá, má pouze předpoklady (chromozomy), během svého života se bude učit a jeho síť se bude vyvíjet.

Neurony mohou mizet a přibývat, stejně jako jejich zakončení a výběžky, které se mohou pohybovat po mřížce a tím měnit množinu neuronů, se kterými bude docházet k výměně signálu. Definice synaptického spojení, stejně jako logika změny tvaru topologie, je řízena příslušnými chromozomy. Obecně můžeme tvrdit, že axon s jeho zakončeními a dendrity s jejich výběžky definují propojovací část sítě, zatímco soma určuje, zda dojde k distribuci signálu prostřednictvím neuronu, ke kterému přísluší.

V rámci knihovny s konkrétním řešením problému jsme pracovali se třemi počátečními zakódováními neuronů do mřížky *(viz. příloha E)*.

4.1.2 Atributy komponent

Ke komponentám neuronu se pojí i několik atributů. Ty definují vlastnosti komponent a představují vstupní proměnné pro chromozomy řídící mechanizmy neuronu. Chromozomy upravují hodnoty atributů komponent, rozhodující a změnách v neuronové síti, které jsou popsaný v následujícím přehledu.

Dodejme, že atributy komponent jsou reprezentovány celočíselnými proměnnými. Všechny celočíselné hodnoty atributů omezíme hodnotou $max = 255$ (celý systém je založen na 8 bitové architektuře). Limit architektury potom označíme jako $base = 256$. Na počátku výpočtu jsou atributy inicializovány náhodně v intervalu $< 0; max >$.

- `health` h - Rozhoduje o replikaci či zániku komponenty. Po úpravě hodnoty h vybrané komponenty chromozomem životního cyklu porovnáváme její velikost se dvěma prahy. Pokud je $h < H_{min}$ ($H_{min} = 0.1max$), dojde k zániku komponenty. V případě, že $h > H_{max}$ ($H_{max} = 0.9max$), dojde k její replikaci. Hodnota h je snižována taktéž systematicky po každém průchodu signálu sítí o konstantu D ($D = 0.05max$). Zajistíme tím, že neaktivní komponenty, u nichž nejsou spouštěny chromozomy životního cyklu obnovující hodnotu h, budou ze sítě po čase odstraněny. Atribut health reprezentuje fakt, že komponenta je využívaná a přenáší signál bez větších ztrát. Její vliv je vidět u chromozomů šíření signálu *(viz. kap. 4.1.4)*.

- `resistance` r - Hodnota resistance souvisí s pohybem komponenty po mřížce. Rozhodující je velikost změny resistance po aplikaci chromozomu životního cyklu. Pokud je velikost změny větší než práh R ($R = 0.3max$), dojde k posunu komponenty. Znaménko změny určí směr pohybu. Vysoká hodnota resistance simuluje dlouhé spojení, které oslabuje šíření potenciálu *(viz. kap. 4.1.4)*.

- `state-factor` sf - Simuluje fakt, že během činnosti mozku nemusí v jeden okamžik pracovat všechny neurony a jejich části. Rozhoduje tedy o tom, zda se vybraná komponenta, resp. chromozom její obsluhy, spustí, či nikoliv. Hodnota 0 implikuje aktivní

komponentu. Pokud je komponenta neaktivní, nemůže šířit potenciál a neúčastní se žádných výpočtů v síti. Na počátku výpočtu jsou hodnoty sf nastaveny náhodně (u somy z množiny $\{0,..,3\}$, u ostatních komponent z množiny $\{0,..,7\}$), velikost čísla udává, kolik cyklů bude komponenta neaktivní. Stejně jako u atributu h dochází ke snižování sf po každém průchodu signálu sítí, abychom síť stimulovali k větší aktivitě.

- `potenciál` p - Jedná se o jednotku informace, kterou síť distribuuje topologií. Postupně se šíří všemi komponentami (v případě, že jsou aktivní) a na základě jeho změn se určuje jejich další aktivita, resp. velikost sf. Pokud dojde k velkému úbytku potenciálu ($\Delta p > P$, kde $P = 0.3max$), nastavíme nenulovou hodnotu sf.
- `weight` w - Určuje, jak (kterými cestami) se bude potenciál šířit sítí. Detailněji je problematika rozebrána u chromozomu nastavení vah *(viz. kap. 4.1.5)*.

Detailní chování sítě při překročení prahových hodnot hodnotou atributu komponenty je popsáno v kapitole věnující se chromozomům životního cyklu *(kap. 4.1.6)*. Nepřímý vliv velikosti hodnoty komponent je popsán v kapitole věnující se chromozomům zpracování vah a šíření potenciálu.

V definované mřížce se tedy budou nacházet neurony složené z více komponent. Každá komponenta má své atributy. Hodnoty atributů jsou zpracovávány chromozomy a tím dochází ke všem dějům v síti, které dále popíšeme.

4.1.3 Vlastnosti chromozomů

Jak jsem již vysvětlovali, chromozomy tvoří výkonnou část jednotlivých komponent. Chromozomy jsou členěny do tří kategorií podle jejich funkce. Můžeme přitom tvrdit, že chromozomy o souhrnné konstantní velikosti můžou generovat libovolně složitou topologii sítě. Každý jeden chromozom odpovídá *CGP* programu. Definujme výchozí vlastnosti takového programu.

Budeme používat zřetězenou strukturu, kde $n_r = 1$ a hodnota $L = n_c$ ($n_c = 30$). Počet vstupů jednotlivých uzlů sítě definujme $n_n = 3$. Funkce proměnných a, b, c, které je možné využít jsou následující (hodnota $base = 256$)

$$f_1 = (ac' + bc) \mod base$$

$$f_2 = (a'c' + bc) \mod base$$

$$f_3 = (ac' + b'c) \mod base$$

$$f_4 = (a'c' + b'c) \mod base$$

Hodnoty negace jsou chápány jako doplněk do celočíselné hodnoty maxima. My budeme uvažovat maximální hodnotu ohraničenou 8 bity (viz. kapitola 4.1.2), proto $a' = max - a$. Z hlediska Boolovy logiky lze pomocí vybraných funkcí rozhodnout, zda je vstup a a b nastaven na hodnotu $true$ či $false$.

Problematická pasáž spočívá zejména v určení vstupu a výstupu některých chromozomů. Počet vstupních hodnot nelze předem deterministicky určit. Je to proto, že síť se dynamicky proměňuje během života jedince. Například chromozom axonálního zakončení, starající se o šíření potenciálu do připojených dendrálních výběžků, může mít na svém vstupu desítky hodnot potenciálů takových zakončení. Proto vstupní hodnoty vzorkujeme konstantou o zvolené optimální velikosti. Chromozom bude mít počet vstupů odpovídající

velikosti konstanty. Velikost konstanty tak určí počet vektorů, které na svých vstupech bude chromozom zpracovávat . Výstupy chromozomu nedefinujeme v rámci zakódování, ale pro zrychlení výpočtu uvažujeme pevně daný seznam náhodně vygenerovaných indexů výstupů CGP programu.

Při mutacích budeme upravovat N_{bit} náhodně vybraných genů. Pokud se jedná o gen reprezentující funkci, vybereme náhodně jinou funkci ze seznamu dostupných funkcí. V případě mutace genu spojení nahrazujeme spojení jiným spojením. Definujme N_{bit} *(*vysvětlení významu proměnných v kap. 2.4.1)

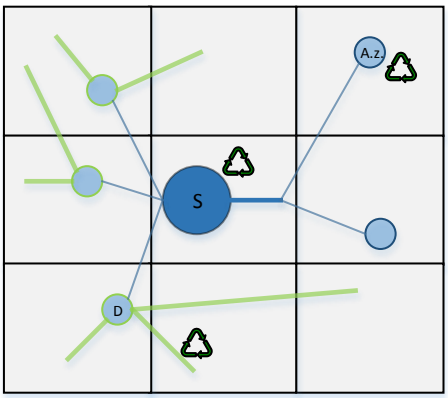
$$N_{\text{bit}} = (n_n + 1)n_c n_{\text{chromozomes}} \frac{\mu}{100}$$

μ ..... `mutační index` $< 0; 100 >$

$n_{\text{chromozomes}}$ ... `počet chromozomů jedince (je pevně daný hodnotou 7)`

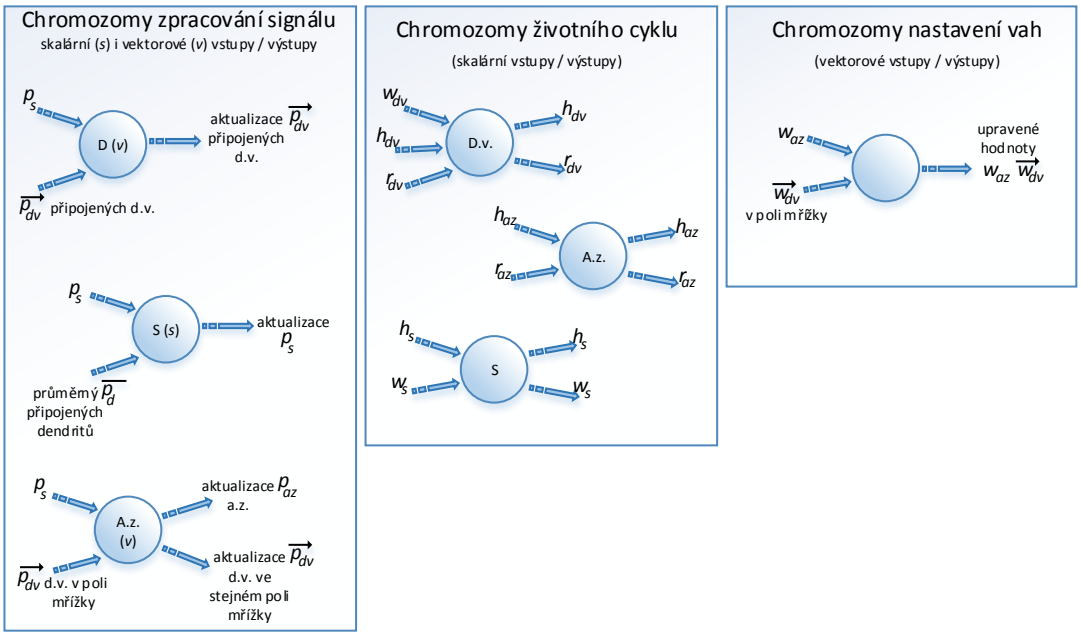
V následujících kapitolách budeme popisovat skupiny chromozomů, které jsou vždy typově shodné. Nejprve si na obrázku 4.3 ukažme jeden neuron s komponentami a nastiňme místa, kde jednotlivé chromozomy budou aplikovány. Na obrázku je vyznačena část mřížky G, do níž jsme inicializovali jeden neuron. Z těla `S` vychází několik dendritů `D` se zeleně znázorněnými výběžky. Axon rozprostírá do mřížky svá zakončení.

První skupina chromozomů zpracovává signály v podobě potenciálů. Tyto chromozomy jsou aplikovány na všech dendritech `D` (vyznačen pro ukázku pouze jedním písmenem `D`), somě `S` a axonálních zakončeních `A.z.`. Další trojice chromozomů ovlivňuje tvar topologie, rozhoduje o replikaci, zániku či změně polohy komponent, nad nimiž je zelený trojúhelník. Jsou to soma, dendrální výběžky a axonální zakončení. Poslední chromozom, který není přímo v schematu zobrazen, je chromozom nastavující dendrální a axonální váhy, rozhodující o směru šíření potenciálu.



Obrázek 4.3: *Mřížka s neuronem a místy působení chromozomů.*

Pro přehlednost ještě ukažme vstupní a výstupní parametry všech skupin chromozomů, jejichž funkce budeme rozebírat v následujících kapitolách.



Obrázek 4.4: *Na obrázku jsou typově rozděleny skupiny chromozomů s popisem jejich vstupních a výstupních hodnot. Jsou použity zkratky: axonální zakončení* `a.z.`*, dendrální výběžek* `d.v.` *a indexace definovaná v 4.1.1. Proměnné odkazují na atributy komponent. Atributy se vztahují vždy k popisované komponentě (případně komponentám ze stejného pole mřížky, pokud je tak specifikováno). Vstupy a výstupy označené jako vektory přísluší komponentám s proměnným počtem, např. dendrální výběžky jednoho dendritu, ty jsou vzorkovány.*

4.1.4 Chromozomy řídící zpracování signálů

Postupně popišme zpracování signálu v jednotlivých komponentách.

Program evolvovaný pro dendrit upravuje potenciál v dendrálních výběžcích na základě potenciálu v somě a potenciálu v aktivních dendrálních výběžcích zvoleného dendritu. Vstupem *CGP* programu jsou navzorkované vektory hodnot potenciálů, výstupem pak aktualizované hodnoty těchto potenciálů. Definujme parametry $\alpha_{dv} = 0.02$, $\beta_{dv} = 0.05$ a $\gamma_{dv} = 0.05$ (vycházíme z [6]), které jsou váhy pro atributy w_{dv}, h_{dv} a r_{dv} aktivních dendrálních výběžků, u kterých aktualizujeme potenciál. Vyšší hodnota h_{dv} a w_{dv} potenciál p_{dv} zvyšuje, resistance naopak. Z biologického hlediska se dá říci, že aktualizovaná hodnota potenciálu v daném dendrálním výběžku je o tolik větší, o co je dendrální výběžek využívanější (hodnoty atributů h_{dv} a w_{dv}) a o tolik menší, o co je vzdálenější (z hlediska polohy v mřížce) od somy (atribut r_{dv}). Nová hodnota potenciálu dendrálního výběžku p'_{dv} (vzniklá úpravou výsledku p_{dv} chromozomálního programu) je

$$p'_{dv} = (p_{dv} + \alpha_{dv} h_{dv} + \beta_{dv} w_{dv} - \gamma_{dv} r_{dv}) \ \ \& \ \ max$$

Všimněme si bitové operace *and*. Je použita jako omezující hodnota dosažitelného maxima. Po úpravě potenciálu dendrálních výběžků určíme hodnotu změny potenciálu v porovnáním s okamžikem před spuštěním chromozomálního programu. V případě, že hladina potenciálu v dendrálním výběžku klesla o zvolenou prahovou hodnotu P (viz. kap. 4.1.2), je hodnota *state-factoru* nenulová a deaktivuje činnost dendrální výběžku na sf_{dv} následujících cyklů (obvykle se hodnota nastavuje maximálně na $sf = 2$). Pokud komponenta zůstává aktivní,

je spuštěn příslušný chromozom životního cyklu (*kap. 4.1.6*).

Chromozom zpracování signálu pro somu, upravující její potenciál, uvažuje jako svůj vstup průměrnou hodnou potenciálu svých dendritů (což je v konečném důsledku průměrná hodnota potenciálu aktivních dendrálních výběžků dendritů) a aktuální potenciál somy p_s. Výsledný potenciál je opět upraven

$$p'_s = (p_s + \alpha_s h_s + \beta_s w_s) \ \& \ max$$

kde $\alpha_s = 0.02$, $\beta_s = 0.05$ [6] se stejnou logikou jako u dendrálních výběžků. Výslednou hodnotu potenciálu p'_s porovnáme s prahovou hodnotou pro výboj ze somy P_{trash} (prahovou hodnotu si každá soma uchovává zvlášť). Hodnota P_{trash} je inicializována na maximální možnou hodnotu, tedy max. Po každém průchodu signálu sítí je automaticky snížena o $0.1max = 2D$ (4.1.2), abychom zvyšovali pravděpodobnost výboje potenciálu. Pokud je potenciál menší než prahová hodnota, může nastat několik možností

- $p'_s < \frac{1}{3}P_{trash}$ - Navýšíme sf_s somy na maximum (pro soma je maximum $sf = 3$), čímž deaktivujeme neuron po delší dobu.
- $\frac{1}{3}P_{trash} < p'_s < \frac{1}{2}P_{trash}$ - Deaktivujeme soma po dobu dvou cyklů.
- $\frac{1}{2}P_{trash} < p_s$ - Deaktivujeme somu po dobu jednoho cyklu, opět nedojde k výboji potenciálu do axonálních zakončení, ale je spuštěn chromozom životního cyklu somy.

V případě, že potenciál překročí práh ($\frac{9}{10}P_{trash}$), dojde k výboji potenciálu. Hodnota prahu, aktuálního potenciálu somy i atributu sf_s jsou nastaveny na maximální hodnoty ($p_s = max$, $P_{trash} = max$, $sf_s = 3$), čímž zajistíme, že neuron bude neaktivní v následujících cyklech algoritmu. To odpovídá klidové fázi neuronu po excitaci. Bude aktivován chromozom životního cyklu somy (*kap. 4.1.6*) a chromozomy zpracování signálu axonálních zakončení, které signál rozšíří k dalším neuronům.

Pokud v somě proběhne výboj, jsou postupně volány chromozomy aktivních axonálních zakončení neuronu, který přísluší této somě. Jako vstupní hodnoty uvažujeme potenciál somy a aktivních dendrálních výběžků, které jsou ve stejném poli mřížky, jako se nachází axonální zakončení. Výstup CGP programu aktualizuje hodnoty potenciálu v dendrálních výběžcích p_{dv} a také v axonálním zakončení (p_{az}), ke kterému se celá procedura vztahuje. To je upraveno stejným výrazem, jako dendrální výběžky při výpočtu jejich hodnot potenciálu chromozomem zpracování signálu dendritu.

$$p'_{az} = (p_{az} + \alpha_{az} h_{az} + \beta_{az} w_{az} - \gamma_{az} r_{az}) \ \& \ max$$

Velikost změny potenciálu v axonálním zakončení ovlivní to, zda bude v následujících cyklech aktivní či nikoliv. Proces je shodný s procesem u dendrálního výběžku. Pokud potenciál axonálního zakončení převyšuje nastavený práh, je ponechán aktivní a je aktivován chromozom životního cyklu. Po výpočtech potenciálu je spuštěn chromozom řídící nastavení vah (*kap. 4.1.5*).

Dodejme, že potenciál celé sítě se v každém cyklu automaticky snižuje o zvolenou hodnotu D u všech komponent. Jak již bylo uvedeno, práh aktivace somy je snížen o dvojnásobek snížení potenciálu somy.

4.1.5 Chromozomy řídící nastavení vah neuronů

Tento chromozom se přímo pojí s dendrálními výběžky a vybraným axonálním zakončením. Jako svůj vstup uvažuje váhy těchto komponent pro jejich společné pole *mřížky*, výstupem

je upravená hodnota w všech zúčastněných komponent. Dendrální výběžek s nejvyšší váhou ve zvolené části mřížky přebírá hodnotu potenciálu právě vybraného axonálního zakončení. Váha rozhoduje o způsobu šíření informací a přímo tak definuje datové kanály sítě.

4.1.6 Chromozomy řídící životní cyklus neuronů

Soma, axonální zakončení a dendrální výběžky využívají také chromozomy životního cyklu. Ty přímo rozhodují o poloze a počtu komponent.

Chromozom životního cyklu dendrálního výběžku (stejně jako axonálních zakončení, kde nepočítáme s w) uvažuje skalární vstup h, r a w a produkuje nově hodnoty h a r. Pokud absolutní hodnota změny *resistance* překročí práh R *(kap. 4.1.2)*, dojde k expandování komponenty, podle znaménka změny je určen směr. Vybíráme místo z osmiokolí aktuální pozice. Nově určená proměnná h rozhodne o zániku komponenty nebo vytvoření nové, s náhodně inicializovanými hodnotami atributů do stejného políčka mřížky. Prahy jsou definovány jako H_{min} a H_{max}.

Životní cyklus somy přímo rozhodne o existenci celého neuronu. Chromozom má skalární vstupy h_s a w_s, které aktualizuje. Pokud je hodnota h_s podprahová ($h_s < H_{min}$), neuron zaniká, pokud nadprahová ($h_s > H_{max}$), inicializuje se nový neuron s náhodnou topologií do jiné části mřížky než stávající.

Za účelem odstranění neaktivních komponent a slabých spojení, po každém kroku sítě (5 cyklech, viz kap. 4.1.9) dojde ke snížení hodnoty w a h všech komponent o konstantu D *(kap. 4.1.2)*. V případě podprahových hodnot se komponenta odstraní.

4.1.7 Vstupy a výstupy sítě

Vstupy sítě jsou reprezentovány axonálními zakončeními, které nepřísluší k žádnému neuronu. Tyto zakončení jsou náhodně rozloženy v *mřížce*. Vstupní signál je duplikován do atributu potenciálu p_{az} axonálních zakončení a je spuštěn chromozom řídící přenos signálu. Následně se již rozbíhá proces distribuce signálu sítí, jak je popsáno v 4.1.8 a 4.1.9.

Výstupy sítě reprezentují náhodně rozmístěné dendrální výběžky, které nepřísluší k žádnému neuronu. Na konci každého kroku výpočtu jsou průměrovány jejich potenciály, které byly nastavovány během výpočtu chromozomy přenosu signálu z axonálních zakončení.

Poloha a počet (bylo použito 5 vstupů a 5 výstupů) vstupních a výstupních komponent zůstává konstantní a nepodléhá vývoji pomocí chromozomů životního cyklu, což pozitivně ovlivňuje celkovou dobu evoluce sítě (více o důvodech použití konstantního počtu vstupů a výstupů v kapitole 6.5).

4.1.8 Popis činnosti jednoho neuronu

Ukažme jeden běh celého neuronu. Uvědomme si, že neuron se může skládat z velkého množství komponent. Všechny komponenty sdílí výše popsaných 7 chromozomů, pomocí nichž se celá síť upravuje.

```
1   for dendrit from seznam všech dendritů neuronu:
2      Aplikuj chromozom zpracování signálu
       na vybraný dendrit (4.1.4)
3      for aktivní dendrální výběžek from dendrit:
4         Spusť chromozom životního cyklu
          pro aktivní dendrální výběžek (4.1.6)
5      end for
6   end for

7   Aplikuj chromozom zpracování signálu pro soma (4.1.4)

8   if je překročen práh pro výboj:
9      for aktivní axonální zakončení from seznam axonálních zakončení:
10        Spusť chromozom zpracování signálu
          pro aktivní axonální zakončení (4.1.4)

11        Spusť chromozom zpracování vah
          pro aktivní axonální zakončení (4.1.5)
12        if axonální zakončení stále aktivní:
13           Spusť chromozom životního cyklu
             pro aktivní axonální zakončení (4.1.6)
14        end if
15     end for
16     Spusť chromozom životního cyklu soma (4.1.6)
17  end if
```

Algoritmus 3: *Činnost jednoho neuronu.*

4.1.9 Popis průběhu algoritmu

V kapitole 4.1.8 jsme popsali netriviální průběh vyhodnocení jednoho neuronu. Průběh celého algoritmu je ohraničen několika zásadními okamžiky, které přímo ovlivňují výsledek výpočtu.

1. `Inicializační fáze` - Na počátku výpočtu jsou náhodně do mřížky inicializovány neurony (maximální počet $N_n = 5$) s náhodně rozmístěnými komponentami. Hodnoty atributů jsou inicializovány z intervalu $\{0, .., max\}$, jedinou odlišností je hodnota $state - factoru$ inicializovaná do intervalu definovaného v 4.1.2 pro dobu neaktivity komponenty. Atributy jsou pro každou operaci zaokrouhleny na celočíselné hodnoty. Chromozomy podléhají také náhodné inicializaci s $n_r = 1$ a $n_c = 30$.

2. `Fáze života jedince` - Do sítě dodáváme vstupy z modelu prostředí M (na vstupní axonální zakončení), ty jsou distribuovány topologií, pomocí výstupů (z výstupních dendrálních výběžků) pak ovlivňujeme prostředí. Podle výsledků během života jedince chromozomy ohodnocujeme pomocí fitness funkce. Obvykle se jeden vstup zpracovává ve více cyklech, vzhledem k možné složitosti a komplexnosti topologie tak, aby bylo možné získat odpovídající výstup sítě. Důležitým faktorem je snižování velikosti sf a p po každém cyklu sítě. Hodnota sf je dekrementována (o 1) , hodnota p snižována konstantně o $D = 0.05max$. Tím se snažíme aktivovat neaktivní komponenty. Zkrácení doby nečinnosti je akcelerováno i strategií snižování hodnoty prahu pro výboj ze somy. Ten se snižuje dvojnásobně tak rychle, jako samotný potenciál somy ($0.1max$). Jeden vstupní signál je zpracováván po dobu 5 cyklů. 5 cyklů tvoří jeden krok sítě, po kterém dojde k dekrementování hodnoty h o D a v případě překročení popsaných prahů z kapitoly 4.1.2, může dojít k odstranění komponenty. Tím se síť zbavuje svých neaktivních částí.

3. `Ukončení života jedince` - V dynamickém prostředí je konec funkčnosti agenta (neuronové sítě vytvořené v mřížce) přímo spojen se splněním jeho úlohy, případně s časovým limitem. Ten zaručí konečnost výpočtu nerozvinutých a ne plně funkčních neuronových sítí.

4. `Ukončení evoluce` - Pod pojem evoluce spadá vývoj chromozomů. Ten je ukončován známými metodami, které např. sledují velikost zlepšení výsledků po sobě následujících generací jedinců. My ohraničíme běh algoritmu konstantním počtem generací, po nichž bude ukončen.

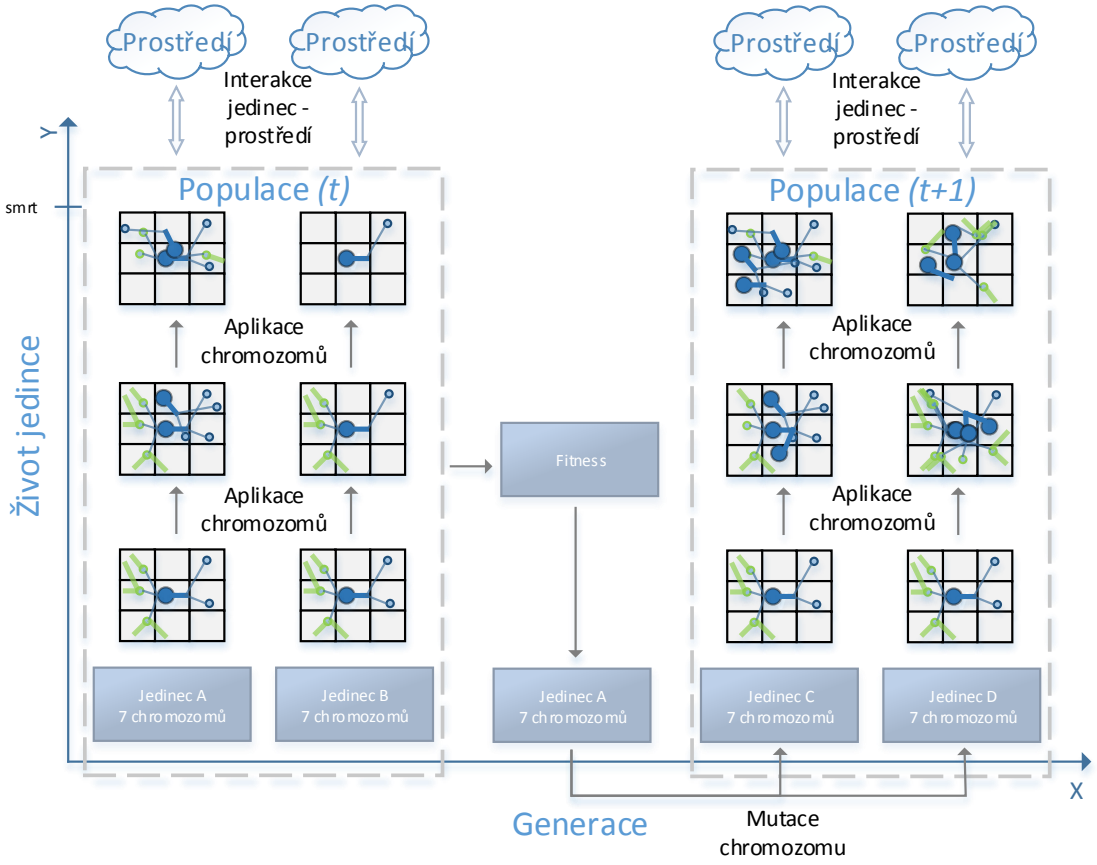
Identifikovali jsem si důležité oblasti algoritmu, uveďme pseudokód algoritmu.

```
    # Inicializace
1   Nastav velikost mřížky ( G_l, G_c )
2   Nastav maximální počet neuronů při inicializaci (N_n)
3   Nastav maximální počet komponent jednoho
    neuronu při inicializaci (N_az, N_d, N_dv)

4   Nastav prahy pro odstranění a replikaci komponent H
5   Nastav konstantu dekrementace hodnot atributů komponent
    po dokončení cyklu (D = 0.05max)

6   Nastav parametry mutací chromozomů (N_bit, μ)
7   Nastav parametry pro CGP chromozomy (n_c, L, Γ)
8   Urči  λ  pro strategii ES(1 + λ)
9   Definuj ukončující podmínky algoritmu E
10  Definuj podmínky ukončení života jedince Death
11  Inicializuj model prostředí M
12  Definuj způsob ohodnocení

13  Do mřížky náhodně vygeneruj počáteční topologii neuronové sítě P
14  Vygeneruj náhodně chromozomy pro 1 + λ jedinců
```

```
    # Výpočet
15  while not E:
16     Obnov P
17     for jedinec from populace:
18        Inicializuj M
19        while  M nedefinuje smrt jedince:
20           Převeď výstup M na vstupní axonální zakončení sítě jedince
21           CGP programy přesuň potenciál ze vstupních
             axonálních zakončení na odpovídající dendrální výběžky
23           for cyklus in {0,...,5}:
24              Identifikuj aktivní neurony
25              for náhodně vybraný neuron from aktivní neurony:
26                 Proveď algoritmus z 4.1.8
27              end for
28              Dekrementuj state-fator všech komponent
                a přehodnoť jejich aktivitu
29           end for
30           Dekrementuj hodnotu h a odstraň nevyužívané komponenty
31           Vykonej akci v M na základě průměrného potenciálu
             výstupních dendrálních výběžků
32        end while
33        Ohodnoť jedince dle výsledků v M
34        if jedinec nejlépe ohodnocen:
35           Uchovej vyvinutou neuronovou síť (S)
36           Uchovej množinu chromozomů s
             počáteční inicializací jedince (CH)
37        end if
38     end for
39     Proveď výběrovou strategii ES(1 + λ) s N_bit mutacemi nad P
40  end while
41  return (S, CH)
```

Algoritmus 4: *Algoritmus průběhu vyhodnocení koevoluce neuronových sítí a genetického programování pro Developmental Network*

Průběh algoritmu znázorněme pro přehlednost obrázkem 4.5.



Obrázek 4.5: *Evoluce sítě Developmental Network.*

Na obrázku můžeme vidět dvourozměrný graf. Na ose X je znázorněn počet generací uplynulých v rámci evoluce. Na ose Y je doba života jednoho jedince. Na počátku výpočtu je inicializována populace $P(t)$ obsahující dva jedince A a B. Oba mají sadu 7 chromozomů. Sady jsou odlišné, jsou náhodně vygenerované a skládají se z CGP programů. Zároveň oba jedinci mají stejnou počáteční síť nainicializovanou do mřížky. Síť jedince začne komunikovat prostřednictvím svých vstupů a výstupů s prostředím popsaným modelem M. Na základě funkce chromozomů a vstupů z prostředí se topologie sítě proměňuje a generuje výstupy. Jakmile se všichni jedinci vyvinou a komunikace s prostředím skončí v bodě smrt, dojde k vyhodnocení fitness funkce chromozomů. Ptáme se, jak byly chromozomy úspěšné při evoluci jednoho jedince během jeho života, resp. jak úspěšně jedinec komunikoval s modelem prostředí a ovlivňoval jeho stav. Nejlepší jedinec je vybrán. Jsou mutovány jeho chromozomy, nikoliv jeho topologie. Vznikají noví jedinci C a D. Jejich topologie se nastaví do stejného počátečního nastavení, jako v předchozí generaci. Noví jedinci mají vzájemně odlišné chromozomy vzniklé mutací chromozomů vybraného nejsilnějšího jedince předchozí populace. Proces se opakuje. Výstupem evoluce jsou chromozomy, pomocí nichž jsme schopni vygenerovat jedince s vysokou fitness hodnotou, případně již vygenerovaná topologie nejlepšího jedince za dobu všech generací.

4.2 Přímé zakódování sítě

Tématu přímého zakódování sítě a použití evoluce pro její vylepšení se věnovalo hned několik návrhů. Přímým zakódováním se řešila problematika nastavení vah spojení neuronů nebo tvar celé topologie. Další experimenty směřovaly k použití nepřímého zakódování, definujícího pravidla tvorby topologie sítě, pro rychlé nalezení tvaru topologie a přímého zakódování k jejímu finálnímu upravení. *X. Yao* ve své práci [16] otestoval několik alternativ evolučních přístupů a zmínil problematiku oddělené evoluce topologie sítě a vah spojení neuronů. V případě separované evoluce dochází k nalezení horších řešení oproti souběžnému evolvování topologie a vah sítě.

Našim záměrem bude pomocí přímého zakódování neuronové sítě do *CGP* chromozomu měnit jak nastavení hodnoty vah, tak tvar a propojení celé topologie, čímž by měl vzniknout robustní a rychle se učící model sítě vycházející z [7]. Způsob zakódování je detailně popsán v kapitole 4.2.1. Návrh zahrne jak dopřednou neuronovou síť v kapitole 4.2.2, tak rekurentní alternativu v kapitole 4.2.3, která má základ v *Jordanově síti* navržené *Michaelem Jordanem*. Konkrétní zakódování řešení problémů z kapitoly 6 lze nalézt na přiloženém *CD (viz. příloha E)*. Popíšeme také upravenou verzi evoluční strategie umožňující nalezení nejlepšího řešení s důrazem na vyšší diverzifikaci genotypů v úvodu evoluce (více v kap. 4.2.5).

4.2.1 Prostředky pro zakódování neuronové sítě

Pro reprezentaci genotypu jsou použity *CGP* chromozomy vycházející z 2.4.1. Topologie je navržena s hodnotou L odpovídající n_c pro maximální možný rozsah propojení orientovaného acyklického grafu, jehož interpretací vznikne neuronová síť.

Každé zakódování neuronu k je definováno sadou genů. Přitom platí, že jednotlivé neurony neuronové sítě jsou surjektivním obrazem uzlů jejich zakódování v chromozomu. Uzly nesou informaci o jejich propojení s okolím, typu použité aktivační funkce a hodnotě výstupu. Všechny uzly mají konstantní počet připojených vstupů odpovídající hodnotě n_n *(viz. kapitola 2.4.1)*. Arita aktivační funkce je jedním z významných činitelů ovlivňujících tvar sítě fenotypu.

Na rozdíl od dříve popsaného *CGP* chromozomu, každé spojení v sobě nese ještě geny s informací o tom, zda je spojení aktivní a jakou má váhu. Uvažujme i.vstup uzlu k referencovaný proměnnou I_{ik}. Jestliže vyjdeme ze známého zápisu zakódování chromozomu, potom nově použitá proměnná pro váhu spojení uzlů (potažmo neuronů ve fenotypu) bude w_{ik}. Pro i platí $i \in \{1, .., n_n\}$. w_{ik} reprezentuje velikost váhy spojení mezi vstupem referencovaným indexem I_{ik} a právě zpracovávaným zakódováním neuronu k. $w_{ik} \in < -1; 1 >$.

Proměnná s_{ik}, $s_{ik} = \{0, 1\}$ definuje ve fenotypu aktivitu spojení mezi neuronem k a zvoleným výstupem uzlu s indexem I_{ik}. Chromozom tímto mechanizmem může nepřímo ovlivňovat počet vstupů neuronu k ve vzniklém fenotypu. Ten má vždy n_n propojení s okolím, ale libovolný počet z těchto propojení může být deaktivován pomocí proměnné s nastavené na hodnotu 0. Aktivita spojení s_{AB} neuronů A a B rozhodne, zda výstupní hodnota neuronu A bude uvažována v rámci argumentu aktivační funkce neuronu B. Mutace může aktivitu spojení během generací měnit.

4.2.2 Zakódování dopředné neuronové sítě

Za pomoci nově definovaného kódování z kapitoly 4.2.1 ukážeme způsob zakódování jednoduché acyklické dopředné neuronové sítě popsané v kapitole 3.3. Neurony ve vrstvě l

mohou být připojeny na neurony z předchozích vrstev nebo na neurony vstupní. Tento fakt v kódování sítě chromozom splňuje se svým parametrem L.

Uvažujme zakódování k.neuronu a ukažme základní odlišnosti od zakódování běžného uzlu CGP chromozomu. Uzel chromozomu reprezentující jeden neuron bude vypadat následovně $((I_{1k}, w_{1k}, s_{1k}), \ldots, (I_{n_nk}, w_{n_nk}, s_{n_nk}), f_{cr})$. Geny označené jako I představují ve fenotypu odkazy na vstupy neuronu, w jejich váhy. Parametr s rozhodne o tom, zda je spojení aktivní. Výstupní hodnota h libovolného neuronu k zakódovaného v r.řádku a c.sloupci chromozomu je pak

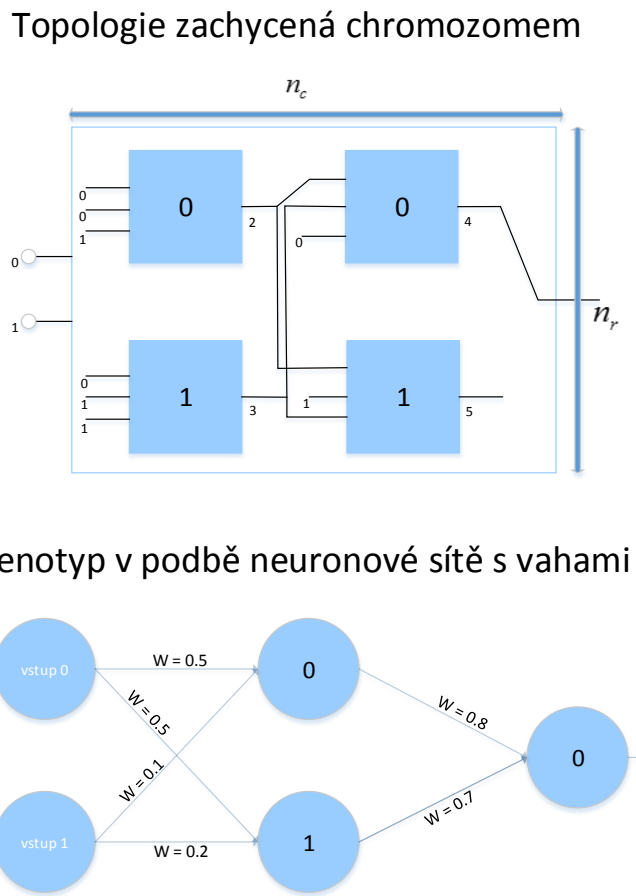
$$h = f^*_{cr}\left(\sum_{i=0}^{i=n_n} s_{ik} w_{ik} I^*_{ik}\right)$$

Použili jsem přitom operátor pro dereferencování indexů výstupů funkcí (resp. indexů do tabulky funkcí) představený v kapitole 2.4.1. Místo indexu výstupu určitého uzlu v rovnici vystupuje přímo hodnota výstupu a místo indexu do tabulky funkcí používáme přímo zvolenou funkci. Dodejme, že u definovaných indexů výstupů $O_1 \ldots O_{n_o}$ z chromozomu jsou pevně zvoleny hodnoty vah a proměnné s na 1. Můžeme je chápat jako ukazatele na vybrané výstupní hodnoty daných uzlů, které výsledky dále neupravují. V některých příkladech z kapitoly 6 ještě výstup sítě budeme upravovat zvolenou funkcí, což můžeme chápat jako důsledek NFL teorému (def. 2.1).

Mějme chromozom

$$\begin{gathered}((0, 0.5, 1), (0, -0.4, 0), (1, 0.1, 1), 0)\\ ((0, 0.5, 1), (1, 0.2, 1), (1, 0.1, 0), 1)\\ ((2, 0.8, 1), (3, 0.7, 1), (0, -0.3, 0), 0)\\ ((2, -0.1, 0), (1, 0.5, 0), (3, 0.7, 1), 1), (4)\end{gathered} \tag{4.1}$$

s konfigurací $L = 2$, $n_r = 2$, $n_c = 2$, $n_i = 2$, $n_n = 3$, $n_f = 2$ a $n_o = 1$. Každý řádek zápisu odpovídá jednomu uzlu chromozomu se třemi vstupy a jednou aktivační funkcí. Indexy aktivačních funkcí směřují na některou z alternativ hyperbolického tangentu a sigmoidy (více v 4.2.4). Na obrázku 4.6 je nejprve vizualizace kompletní topologie genotypu a následná transformace ve fenotyp dvouvrstvé dopředné sítě (postupné řazení uzlů do topologie probíhá stejně jako v případě interpretace chromozomu u kartézského genetického programování v kapitole 2.4.1). Můžeme vidět, jak přepínač s ve výsledku ovlivňuje tvar neuronové sítě, kdy některá spojení definovaná v zakódování ve fenotypu nevidíme (například uzel, jehož výstup je indexován indexem 2 má deaktivovaný 2. vstup). Je to z toho důvodu, že jejich hodnota genu $s = 0$.



Obrázek 4.6: *Vizualizace kompletní topologie dopředné sítě z chromozomu 4.1 a její redukce na fenotyp.*

4.2.3 Zakódování rekurentní neuronové sítě

Evolvovaný model neuronové sítě vychází z modelů jednoduchých rekurentních neuronových sítí. V experimentech je použit přepracovaný model *Jordanovy neuronové sítě.*

Definice 4.2. *`Jordanova neuronová síť` - Síť má charakter dopředné třívrstvé neuronové sítě obohacené o tzv. stavové jednotky, navrženou Michaelem Jordanem [5]. Počet stavových jednotek je stejný jako počet výstupních neuronů. Každá stavová jednotka je propojena právě s jedním výstupním neuronem a zprostředkovává jeho výstup rekurentně jako další vstup neuronové sítě. Každá stavová jednotka má také rekurentní spojení sama na sebe.*

Tento model upravme pro potřeby spolupráce *CGP* a neuronových sítí. Pro každý výstup sítě bude existovat stavová jednotka s výše popsanými vlastnostmi. Stavové jednotky budou mít navíc implicitně určenou aktivační funkci odpovídající hyperbolickému tangentu. Stavové jednotky proto budou představovat přídavné vstupy neuronové sítě. Váha spojení a hodnota přepínače s mezi výstupem neuronové sítě a stavovou jednotkou, stejně tak jako hodnota těchto parametrů u rekurentního spojení stavové jednotky samé na sebe, bude na hodnotě 1. Hodnota těchto parametrů nebude podrobena mutacím chromozomu zakódované neuronové sítě.

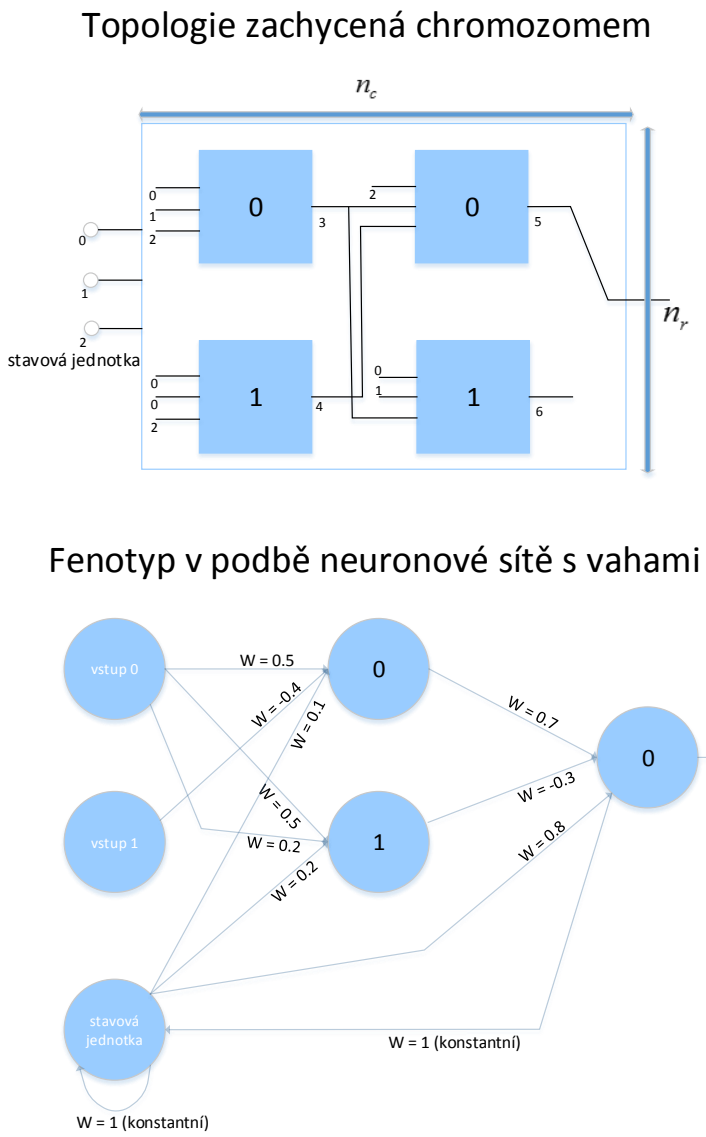
Nyní ukažme, jak bude vypadat zakódování chromozomu neuronové sítě. Počet vstupů chromozomu n_i' bude navýšen o počet stavových jednotek, tedy $n_i' = n_o + n_i$. Indexy I, které budou referencovat uzly z množiny $\{n_i + 1, \ldots, n_o\}$, nebudou odkazovat v chromozomu přímo na vstupy, nýbrž na výstupy stavových jednotek. V zápisu chromozomu, stejně jako v případě vstupních bodů, nejsou přímo stavové jednotky zmíněny, pouze se na jejich výstupní hodnoty odkazujeme. Arita funkcí je zvýšena úměrně počtu přibylých vstupů.

Při počáteční inicializaci chromozomu vždy jeden vstup uzlu v prvním sloupci chromozomu odkazuje na stavovou jednotku s aktivovaným přepínačem s. Veškerá ostatní propojení zbylých uzlů jsou generována náhodně. Mutace mohou spojení dále libovolně proměnit. Tato inicializace je použita proto, aby alespoň na začátku jakéhokoliv výpočtu měla vygenerovaná síť podobu sítě rekurentní. Mutace totiž mohou způsobit, že evolvované chromozomy budou v konečném důsledku generovat síť dopřednou a to v případě, že stavové jednotky budou odpojeny od ostatních uzlů v chromozomu, nebo jejích spojení bude mít parametr $s = 0$.

Ukažme si jeden konkrétní příklad chromozomu a neuronové sítě. Formát zakódování jednoho neuronu je shodný s popisem zakódování neuronu dopředné sítě z kapitoly 4.2.2. Parametry chromozomu bude $L = 2$, $n_r = 2$, $n_c = 2$, $n_i = 3$, $n_n = 3$, $n_f = 2$ a $n_o = 1$. Interpretací chromozomu může vzniknout až dvouvrstvá síť, vstup sítě bude tvořen 2 standardními vstupy a jedním výstupem stavové jednotky. Ukažme si jednu z možných počátečních konfigurací. V příkladu nedefinujeme přesně charakter aktivačních funkcí, na které se odkazujeme. V sekci věnované experimentům to budou funkce popsané v kapitole 4.2.4.

$$\begin{gathered}((0, 0.5, 1), (1, -0.4, 1), (2, 0.1, 1), 0) \\ ((0, 0.5, 1), (0, 0.2, 1), (2, 0.2, 1), 1) \\ ((2, 0.8, 1), (3, 0.7, 1), (4, -0.3, 1), 0) \\ ((0, -0.1, 0), (1, 0.5, 0), (3, 0.7, 1), 1), (5)\end{gathered} \tag{4.2}$$

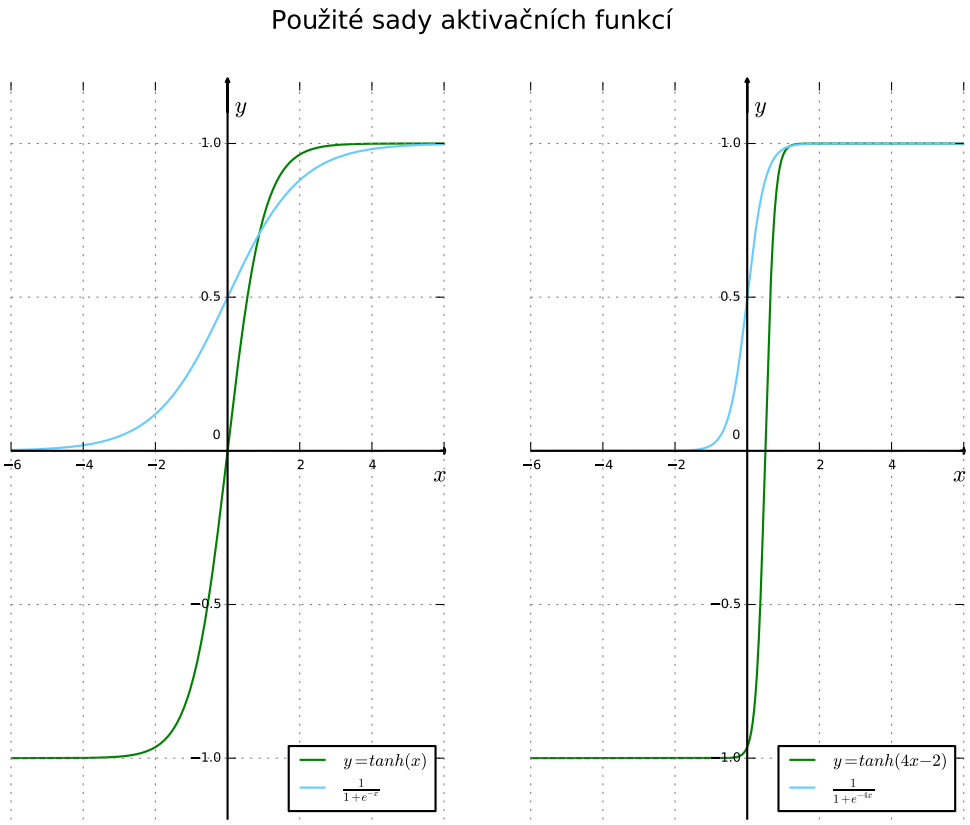
Index 2 v chromozomu odkazuje na uzel, který reprezentuje stavovou jednotku. Jejím vstupem je její výstup a výstup uzlu 5.



Obrázek 4.7: *Vizualizace kompletní topologie rekurentní sítě z chromozomu 4.2 a její redukce na fenotyp.*

4.2.4 Aktivační funkce

V kapitole 3.2 jsme rozebírali důvody použití vybraných aktivačních funkcí. V evolvovaných modelech obecně poskytujeme více alternativ aktivačních funkcí pro jednotlivé neurony. Mutace potom můžou u jednotlivých neuronů aktivační funkce měnit. Pro modely popsané v části experimentů 6 jsme volili vždy dvojice sigmoidy a hyperbolického tangentu.



Obrázek 4.8: *Dvě sady použitých aktivačních funkcí.*

Rozdíl mezi sadami z obrázku 4.8 je dán požadavky na vstupy a výstupy sítě. Pro sítě, jejímž vstupem a výstupem jsou bipolární hodnoty používáme sadu neupravených funkcí tangentu a sigmoidy. Tak to bude například v experimentech s vyvažováním tyče v kapitole 6.2. Druhou sadu s výhodou používáme v případě vstupů z intervalu $< 0; 1 >$. Takové sítě budou typické pro problém detekce buněk rakoviny v kapitole 6.1 nebo problém pohybu agenta v bludišti z kapitoly 6.3.

V případech, kdy požadujeme obor hodnot neuronové sítě pouze v intervalu $< 0; 1 >$, je běžným postupem, že původní výstup sítě je transformován pomocí sigmoidální funkce. Naopak pro případy vyžadující bipolární hodnoty se jako finální úprava osvědčila funkce hyperbolického tangentu. V konkrétních případech na to čtenáře upozorníme.

4.2.5 Mutace a úpravy evoluční strategie

V případě kartézského genetického programování jsou mutace jedinou technikou, jak diverzifikovat populaci a vytvořit nové jedince. Jmenujme skupiny genů, kterých se mutace dotýkají

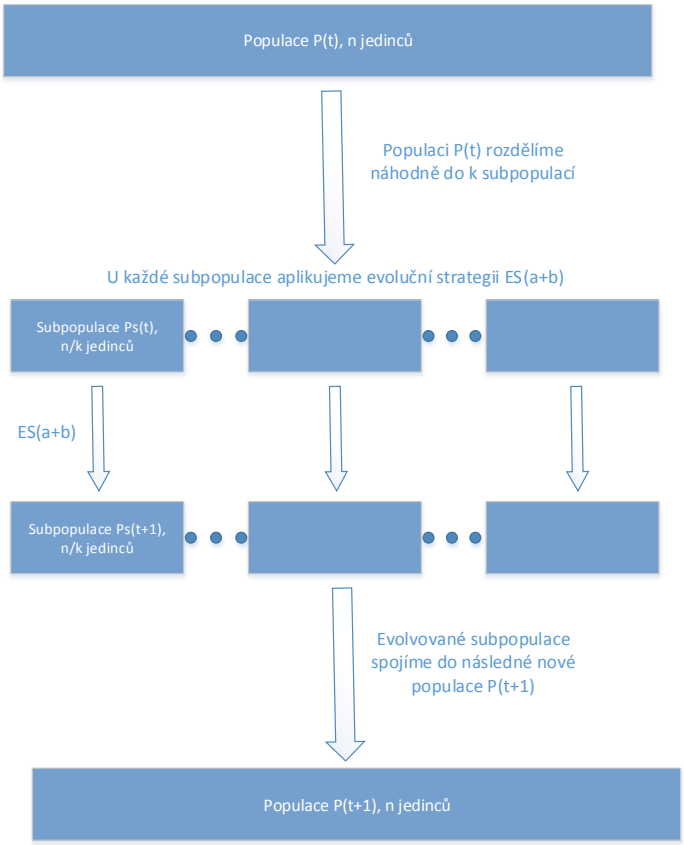
- `Geny propojení uzlů` - Mutace mění hodnotu indexu na jinou přípustnou hodnotu připojeného uzlu s ohledem k hodnotě L.

- `Geny aktivity propojení uzlů` - Při mutaci je aktivní spojení deaktivováno a naopak.
- `Geny vah propojení uzlů` - Vygenerujeme novou váhu náhodně z intervalu $< -1; 1 >$.
- `Geny aktivační funkce` - Je vybrána jiná aktivační funkce z definované množiny.

Mutace postihuje $\mu\%$ genů (v provedených experimentech $\mu \in \{10, 20\}$). Geny jsou měněny vždy na stejný typ, nestane se proto, že by se např. gen spojení uzlů změnil na gen definující typ aktivační funkce. Iniciální populace, stejně jako populace v průběhu generací, mají několik parametrů neměnných mutací. Jsou jimi L, n_r, n_c, n_i, n_n, n_f a n_o.

Každý jedinec proto může vytvořit jen určitou ohraničenou množinu svých potomků danou možnostmi mutace a velikostí populace. Pokud by evoluce vyžadovala vytvoření několika generací jedinců s nižším ohodnocením, nežli má rodič tak, abychom nakonec dostali jedince s ohodnocením vyšším, s běžně používanou evoluční strategií $ES(1 + 9)$ bychom nemuseli uspět. Limitovala by nás právě ohraničená množina možných potomků rodiče, případně typ evoluční strategie. Za účelem většího rozrůznění genotypů a výsledného chování fenotypů uveďme některé další techniky, která mohou podporovat mutační operátory při evoluci. Tyto techniky budou zařazeny do experimentů kapitoly 6 s cílem zjištění jejich vlivu při evoluci.

- `Použití strategie` ES$(n + m)$ - Udržováním n nejlepších jedinců se nám může podařit v populaci udržet více divergentních směrů vývoje nejlepšího řešení.
- `Inicializace populace s jedinci s odlišnými parametry` n_r `a` n_c - Zmíněné parametry určují limitu velikosti výsledné neuronové sítě. V případě hledání nejlepšího řešení vytváříme populaci kombinací genotypů, jejichž parametry nejsou stejné, ale jsou náhodně voleny kolem vybrané hladiny.
- `Použití evoluční strategie` ES$(n + m)$ `v rámci subpopulací` - Spojením dvou předchozích přístupů dostáváme použití evoluční strategie v subpopulacích. V případě, že implicitně vytváříme rozdílně velké genotypy (ač se jedná o zvolený interval), našim cílem je, aby genotypy, které zkonvergují do lokálního extrému, nevyhladily z populace méně kvalitní řešení, které mohou konvergovat po delším čase k lepším výsledkům. Zpomalení expanze zajišťuje náhodná distribuce jedinců do subpopulací. Až v subpopulacích probíhá krok evolučního algoritmu. Poté se subpopulace sloučí do populace v následující generaci (více na obrázku 4.9).



Obrázek 4.9: *Ukázka náhodného vygenerování subpopulací z výchozí populace, provedení mutací podle ES(a+b) a konečně spojení subpopulací do nové populace.*

- `Více výstupů z chromozomu` - Každý výstup chromozomu reprezentuje vygenerovanou posloupnost operací. Při více výstupech se zjemňuje vliv mutace genů výstupu. U klasifikačních úloh je použito více výstupů, jejichž hodnoty jsou průměrovány.

4.2.6 Popis průběh algoritmu

V kapitole 6 jsou popsány 3 modely (model pro vyvažování tyče na vozíku, model bludiště a model definující trénovací a testovací množinu pro klasifikaci rakoviny), na kterých bude testována technika koevoluce *CGP* a neuronových sítí. Model vždy představuje zdroj dat a ohodnocení pro jedince, v našem případě zakódované neuronové sítě. Model také určí, zda jeho komunikace s neuronovou sítí má či nemá být ukončena. V případě určování rakovinných buněk je doba komunikace ovlivněna velikostí datové sady, u vyvažování tyče definujeme maximální počet kroků vyvažování. V případě bludiště se jedná o energetickou funkci agenta. Ukažme obecný průběh algoritmu s důrazem na zakomponování modelu prostředí do výpočtu. Princip odpovídá obecnému algoritmu 2.

```
1   inicializuj čas t
2   inicializuj testovaný model M
3   definuj konfiguraci chromozomů jedinců populace
4   náhodně inicializuj a ohodnoť s M novou populaci P(t)

5   while není splněna definovaná ukončující podmínka algoritmu:
6      for jedinec C from P(t):
7         inicializuj M
8         převeď genotyp C na fenotyp F (neuronovou síť)
9         while M nedefinuje konec komunikace:
10            výstupní hodnoty M přiveď na vstupy F
11            zpracuj vstupy F
12            výstup F přiveď na vstup M
13         end while
14         ohodnoť C pomocí M
15     end for
16     t' = t+1
17     aplikuj vybranou evoluční strategii
       a vytvoř novou populaci P(t') z P(t)
18     t = t'
19  end while
20  return nejsilnější jedinec z P(t)
```

Algoritmus 5: Algoritmus začlenění modelu do evolučního výpočtu.

Kapitola 5

Popis implementace

Pro ověření teoretické části návrhu koevoluce genetického programování a neuronových sítí vznikla knihovna implementující hlavní části návrhu v jazyce `C++`. Přiložené *CD (příloha E)* obsahuje kromě samotné knihovny také sadu `Bash` a `Python` scriptů, pomocí kterých byly prováděny statistické výpočty a vykreslení grafů.

Řešení bylo otestováno na několika 64*bit* linuxových distribucích, zejména pak na operačních systémech `Fedora` a `Debian`.

Interpretace `Python` scriptů je v prostředí `Fedora` prováděno za pomoci balíčku `python-2.7.5-11.fc20.x86_64`. Pro matematické výpočty jsou použity optimalizované matematické knihovny `NumPy`(`numpy-1.8.0-4.fc20.x86_64`) a `SciPy` (`scipy-0.12.1-1.fc20.x86_64`). Grafické výstupy zprostředkovává knihovna `MatPlotLib` (`python-matplotlib-1.3.1-3.fc20.x86_64`), která je alternativou prostředí MATLAB.

Kompilace zdrojových kódů jazyka `C++` byla v operačním systému `Fedora` prováděna s balíčky `gcc-c++-4.8.2-7.fc20.x86_64` a `gcc-4.8.2-7.fc20.x86_64`. V návrhu byl kladen důraz na využití metaprogramování, `STL` šablon a výhod objektového návrhu [10]. Jmenujme jen hlavní třídy knihovny a způsob jejich implementace, podrobná dokumentace je vygenerována automaticky pomocí programu `Doxygen` a je přiložena na *CD*.
Celá implementace knihovny je zapouzdřena ve jmenném prostoru `namespace cgpdnn`, což usnadňuje její import do dalších projektů. Obsahuje jak implementaci *Developmental Network*, tak přímých alternativ zakódování neuronových sítí.

Nejprve popišme strukturu návrhu *Developmental Network (více v 4.1)*.

- `class Random` - Třída představuje návrhový vzor *singleton*. Implementuje v sobě kongruentní generátor náhodných čísel inicializovaný pomocí časového seedu. Její instance je sdílena pro celý jmenný prostor.

- `class CgpEngineInterface` - Abstraktní virtuální třída poskytuje rozhraní hlavní výpočetní jednotce *CGP*. Třídy dědící toto rozhraní mohou být použity v šablonách chromozomů (`template<class Engine> class Chromosome*`) jako šablonový typ.

  - `class CgpEngine` - Třída implementuje chromozom kartézského genetického programování *(více v 2.4)*. Nedekódovaný chromozom uchováváme v paměťově náročné *STL* šabloně `vector` s názvem `cgpStructure`. Virtuální metodou `activate` chromozom dekódujeme dle algoritmu z [9]. Vzhledem k počtu propojení N se jedná o operaci spadající do třídy složitosti $O(N)$. Dekódovaný chromozom je uchováván především v podobě ukazatelů na původní strukturu, což snižuje paměťové i výpočetní nároky implementace.

- `template<class Engine> class ChromosomeLifeCycleInterface`
- `template<class Engine> class ChromosomeSignalProcessingInterface`
- `template<class Engine> class ChromosomeWeightProcessingInterface`
  Skupina abstraktních virtuálních šablonových tříd tvořících rozhraní pro množinu tříd popisující chromozomy.
  - `template<class Engine> class ChromosomeLifeCycle`
  - `template<class Engine> class ChromosomeSignalProcessing`
  - `template<class Engine> class ChromosomeWeightProcessing`
    Šablonový typ je instanciován pomocí třídy dědící z třídy `CgpEngineInterface`. V našem řešení třídou `cgpEngine`, která představuje výpočetní jednotku chromozomu *(více v kapitole 4.1.3)*. Třídy `Chromosome*` mají přístup k privátním datům třídy `NeuralTopology`, jejíž datové struktury na základě výpočtů `cgpEngine` upravují.
- `class CreatureInterface` - Rozhraní třídy `Creature`.
  - `class Creature` - Třída zapouzdřuje jednoho jedince populace, který ve své instanci obsahuje objekt třídy `NeuralTopology` a sadu svých chromozomů. Metoda `run` představuje z hlediska časové složitosti nejslabší místo celého algoritmu, jelikož v každém kroku probíhá vyhodnocení nad všemi komponentami v mřížce.
- `class NeuralTopologyInterface` - Abstraktní virtuální třída definující rozhraní třídy `NeuralTopology`
  - `class NeuralTopology` - Třída představuje datový sklad pro mřížku a komponenty do ní vložené *(více v 4.1.1)*. Datové struktury a metody s nimi pracující jsou v privátní části třídy. Instance tříd `Creature` a `Chromosome*` jsou deklarovány jako `friend`. Data jsou tak přístupná pouze omezené množině tvořené instancemi spřátelených tříd.
- `class EnvironmentMaze1` - Třída definuje model bludiště z 6.5.

Knihovna obsahuje i třídy reprezentující přímé zakódování neuronových sítí:

- `class CgpRNN` - Třída představuje upravenou variantu třídy `CgpEngine` pro zakódování rekurentní neuronové sítě do `CGP` chromozomu. Jedná se zejména o přidání vrstvy se stavovými jednotkami a rozšíření datových struktur o proměnné vah atd. *(více v kapitole 4.2.3)*.
- `class CgpFNN` -Třída `CgpFNN` představuje upravenou variantu třídy `CgpEngine` pro zakódování dopředné neuronové sítě do `CGP` chromozomu *(více v kapitole 4.2.2)*.
- `class EnvironmentMaze2` - Třída modelu bludiště popsaného v kapitole 6.4.
- `class PoleBalancing` - Třída modelu vyvažování tyče popsaného v kapitole 6.2.
- `class CancerLoader` - Třída modelu vytvářející trénovací a testovací množinu z datasetu s rakovinnými buňkami.

Všechny třídy popisující modely mají metody `input` a `output`, pomocí nichž komunikují s okolím. Starají se obvykle i o správné ohodnocení jedince a ukončení vyhodnocení jednoho jedince.

Knihovna také definuje sadu výjimek, pomocí nichž jsou řešeny nesprávné inicializace instancí popsaných tříd, včetně těch šablonových a další nestandardní chování (například při ukládání a načítání uložených konfigurací).

Důležitou součástí tříd kódujících řešení je také možnost uložení a znovupoužití vygenerovaných zakódování sítí, které lze ve formátu *csv* uchovat. Děje se tak díky přetíženým operátorů `operator<<()` a `operator>>()`. Konkrétní ukázky zakódování neuronových sítí jsou přiložený ve zdrojových kódech *(viz. příloha E)*.

Stejně jako zakódování sítě v chromozomu lze uložit počáteční zakódování mřížky *Developmental Network*.

Poslední část implementace, kterou zmíníme, jsou předpřipravené funkce v souboru `main.cpp`. Jsou jimi `poleBalancingFNN`, `poleBalancingRNN`, `cancerFNN`, `mazeBarrierRNN`, `mazeCycleRNN` a `mazeDN`. Tyto funkce implementují standardní koevoluční algoritmy pro získání řešení popsaných v kapitole 6. Při spuštění připraveného dema na přiloženém *CD* si lze vybrat již vygenerovanou neuronovou síť pomocí těchto funkcí a prohlédnout si její zakódování do chromozomu, případně otestovat její chování na modelu prostředí.

Kapitola 6

Experimenty

Experimenty byly provedeny nad třemi různými modely. Jsou jimi detekce buněk rakoviny prsu *(*kap. 6.1), problém vyvažování tyče *(kap. 6.2)* a problém agenta v bludišti *(kap. 6.3)*. Pro jednotlivé úlohy jsme testovali metody přímého i nepřímého zakódování neuronové sítě. V každé kapitole je konfigurace výpočtu popsána detailně. Jsou také diskutovány vlastnosti modelů, zejména jejich rozhraní. Každý jedinec s modelem komunikuje. Model definuje konec komunikace, během které je jedinec ohodnocen. Průběh komunikace nebyl dosud v žádné kapitole zmíněn detailně, jelikož je specifický pro konkrétní modely. V algoritmu 5 je zobecněný průběh komunikace pro přímé zakódování sítě, jemuž se zejména budeme věnovat.

Způsob provedení mutací, jakožto realizace evolučních strategií pro jednotlivá zakódování, jsou uvedeny v odpovídajících teoretických kapitolách *(4.1.3, 4.2.5, 2.2)*. V teoretických kapitolách jsou obecně popsány i průběhy algoritmů evoluce s přímým *(5)* i nepřímým zakódováním *(4)* .
Experimenty se zaměřují na několik dílčích cílů

1. Závislost rychlosti a přesnosti učení na vybraných parametrech (n_n, n_c, mutace, evoluční strategie).
2. Míra přeučování při učení nad klasifikačními úlohami.
3. Robustnost sítí.
4. Míra ovlivnění vlastností evolvovaných neuronových sítí při změnách jejich ohodnocení.

Pro přehlednost je v příloze A uvedena tabulka nejčastěji použitých symbolů.

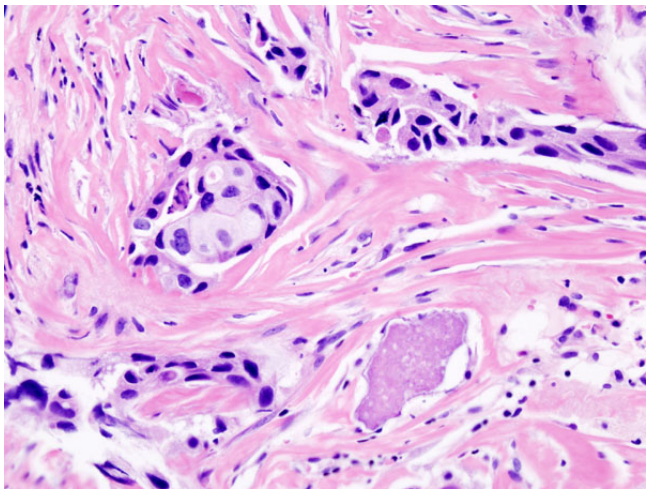
6.1 Testování buněk karcinomu prsu

Karcinom prsu patří k velmi nebezpečným rakovinným onemocněním. Obvykle postihuje ženy po 55 roce věku. Statisticky v České republice takto onemocní 105 lidí ze vzorku 100000 obyvatel, přitom úmrtnost je 33% [1].

V dnešní době je použito několik metod klasifikace, zda nalezený tumor je maligního či benigního původu. Mezi základní techniky patří biopsie prsu. Při ní je část podezřelé tkáně

[1] Zdrojem jsou epidemiologická data rakovinných onemocnění na www.svod.cz.

odebrána a biochemicky analyzována. Bezbolestnou metodu představuje mamografické vyšetření, kdy je tkáň ozářena paprsky X a vizuálně testována. Prokazatelnost nádorového onemocnění touto metodou není 100%.



Obrázek 6.1: *Skirhotický intraduktální karcinom.*

6.1.1 Popis modelu a jeho použití

Model je v tomto případě reprezentován trénovací a testovací množinou dat. Pro naše účely použijeme data z *Wisconsinské datové sady*[1]. Datová sada vznikala v letech 1989 až 1991 ve fakultní nemocnici ve Wisconsinu pod vedením *Dr. Williama H. Wolberga*. Obsahuje celkem 444 benigních a 239 maligních záznamů. Buňky podezřelé tkáně byly získány méně invazivní metodou biopsie nazvanou *Fine Needle Apiration (dále FNA)*. Při této metodě dojde k odběru tkáně pomocí dlouhé tenké jehly zavedené ke zkoumanému místu. U takto vybraných buněk bylo popsáno 9 charakteristických atributů (např. celistvost buněčného obalu, vlastnosti buněčné jádra atp.). Atributy byly ohodnoceny na stupnici od 1 do 10. Je také uvedeno, zda se jednalo o buňky benigní (0), či maligní (1).

Prostá klasifikace buňky podle jednoho atributu není možná. Našim cílem je tedy evolvovat dopřednou neuronovou síť, která bude schopna na základě kombinace uvedených atributů správně klasifikovat rakovinnou povahu buňky. V případě evoluce sítě tedy model při ohodnocení sítě poskytuje pouze data trénovací množiny (při zkoumání vlastností sítě, míry přetrénování a robustnosti sítě se jedná i o testovací množinu) v podobě vektorů délky 9 s atributy popisujícími odebranou buňku.

Hodnoty atributů vzorku jsou přivedeny na vstupy fenotypu, ten je vyhodnotí. Hodnoty všech výstupů sítě (10) jsou pro každý vzorek průměrovány a porovnány s prahem $\frac{1}{2}$. Pokud je průměrná hodnota výstupu $x < \frac{1}{2}$, definujeme výstup sítě jako 0 (benigní forma rakoviny), v opačném případě je výstupní hodnota sítě 1 (maligní forma rakoviny). Výstup sítě představuje vstupní hodnotu modelu. Ten porovná predikovaný typ rakoviny se správným řešením. Pokud dojde ke shodě, inkrementuje ohodnocení sítě f_v o 1 (počáteční ohodnocení každé sítě $f_v = 0$), v opačném případě ohodnocení zůstane stejné. Ohodnocení jedné sítě provedeme vždy pro celý trénovací set, odpovídá sumě správně určených tréno-

[1]Datová sada pro testování rakovinných buněk je k nalezení na http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

vacích vzorků. Takto ohodnocené jedince podstupujeme vybrané evoluční strategii a celý algoritmus tak může konvergovat k optimálnímu řešení.

Pro natrénování a testování sítě použijeme dvě datové sady, sadu A a B. Obě vznikly rozdělením *Wisconsinské datové sady* na dvě disjunktní podmnožiny, trénovací a testovací. V sadě A trénovací i testovací set obsahuje 200 záznamů. V obou setech je stejný počet maligních a benigních vzorků, tedy 100. Sada B pokrývá celou *Wisconsinskou datovou sadu*, ale neobsahuje stejný počet vzorků pro jednotlivé formy rakoviny. Testovací set obsahuje celkem 342 vzorků, 121 maligních a 221 benigních. Trénovací set obsahuje 341 vzorků, 118 maligních a 223 benigních.

6.1.2 Konfigurace výpočtu

Při výpočtech bude použita $ES(5+5)$ a $ES(1+9)$ s různými nastaveními tvaru genotypu. Váhy, přepínače aktivity spojení, propojení uzlů a aktivační funkce ze sady aktivačních funkcí, jsou v chromozomu inicializovány náhodně, definujeme několik stupňů arity aktivačních funkcí. Chromozomy mají vždy $n_r = 1$, velikost n_c je pro jednotlivé experimenty měněna. Aktivační funkce jsou vybírány mezi $y = tanh(x)$ a $y = \frac{1}{1+e^{-x}}$, každý výstup z chromozomu je upraven sigmoidou *(obr. 4.8)*. Důvodem je fakt, že výstup je očekáván právě v intervalu $< 0; 1 >$. Chromozomy mají vždy 9 vstupů, pro každý z atributů jeden a 10 výstupů.

Pro každé nastavení chromozomu provedeme 5 testů ohraničených 11000 generacemi evoluce, ze kterých budeme interpretovat statistické údaje. Síť s nejlepšími výsledky natrénovanou na trénovací množině spouštíme na množině testovací, abychom ověřili její robustnost a další vlastnosti.

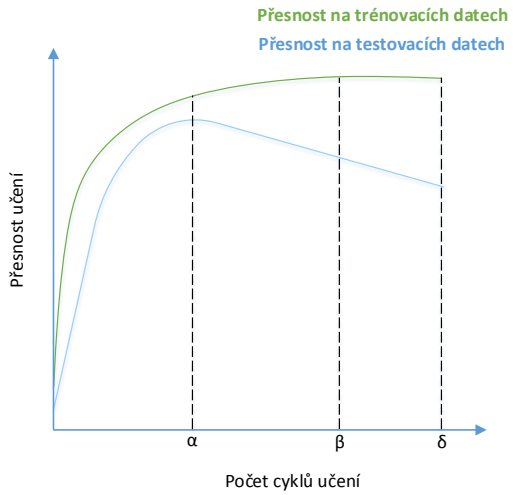
6.1.3 Vyhodnocení měření

Na obrázku 6.2 je znázorněn běžný průběh učení genotypu nad trénovacími daty a jeho vyhodnocení nad daty testovacími. Na ose y je vynesena přesnost nejlepšího jedince pro danou generaci (o ohodnocení jedinců více v kapitole 6.1.1). Na ose x, která reprezentuje stav, ve kterém se nachází evoluce, jsou vyznačeny tři body, které budou hrát v našich experimentech zásadní roli.

- `Bod` α - Představuje první výskyt maxima přesnosti na testovací množině s odpovídající nejvyšší možnou přesností na množině trénovací. V dalších generacích už se přesnost sítě na testovací množině nezvýší, zůstane konstantní nebo dojde k přetrénovaní sítě. V podstatě se tedy jedná o nalezení bodu časného zastavení *(více definice 3.1)*.

- `Bod` β - Určuje první výskyt maximální hodnoty přesnosti na trénovací množině a k ní příslušné maximum na množině testovací.

- `Bod` δ - Definuje hodnoty přesností na konci evoluce.

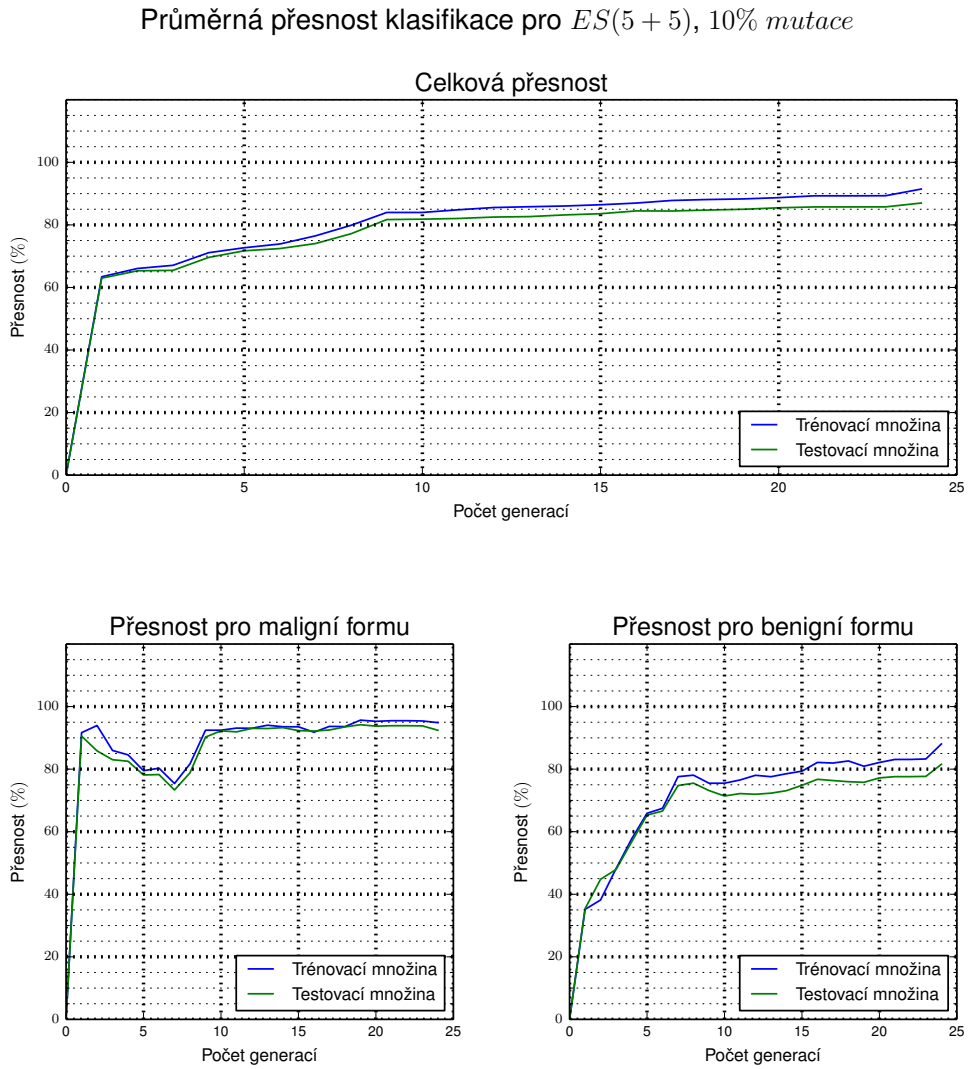


Obrázek 6.2: *Ukázka typického vývoje přesnosti klasifikace fenotypu po daném počtu iterací evolučního algoritmu.*

Pomocí těchto tří bodů lze určit zda a jak se síť dokáže učit, případně kdy dochází k přeučování.

Postupně zmapujme fáze učení. Podotkněme, že pro grafy jsou vybírány výsledky vzniklé použitím $ES(5+5)$. Je to proto, že v případě populace s jedním rodičem může docházet v následných generacích k alternacím mezi dvěma či více rodiči. To plyne z principu, který tvrdí, že pokud máme dva rodiče se stejným ohodnocením na trénovací množině, jako rodiče do další generace uvažujeme novějšího z nich *(jedná se o základní vlastnost genetického programování popsaná v algoritmu 2)*. $ES(5+5)$ tyto alternace dokáže eliminovat a zároveň podat relevantní informaci například o vyhodnocení nejlepšího jedince na testovací množině pro zvolenou populaci. To bude například v případě, že více jedinců má stejnou přesnost na trénovacích datech, ale pouze jeden z nich má nejvyšší přesnost na datech testovacích.

První fáze trénování probíhá do bodu α. Bez rozdílu na konfiguraci velikosti chromozomu či způsobu rozrůznění populace dochází k rychlé konvergenci algoritmu. Na obrázcích 6.3 a B.1 můžeme vidět průměrný vývoj přesnosti na trénovací a testovací množině během evoluce. Grafy v sobě průměrují výsledky všech jedinců pro danou míru mutace a evoluční strategii. Zvlášť jsou zobrazeny vývoje přesnosti na maligní a benigní testovací sadě.



Obrázek 6.3: *Rychlost konvergence při nízkých mutacích. Uvažujeme maximální trénovací přesnost a příslušnou maximální testovací přesnost.*

Průběhy křivek testovací a trénovací množiny kopírují velmi podobnou trajektorii. Během 20 generací, kdy se vytvoří mutacemi při strategii $ES(5+5)$ nejvýše 105 unikátních jedinců, dosahuje výpočet přesnosti více než 80% na trénovací i testovací množině. Rychlost konvergence je pozitivním faktorem i vzhledem k častým časovým problémům konvenčních metod učení složitějších topologií neuronových sítí *(viz. kap. 3.6.1)*. Objektivně musíme konstatovat, že pro definitivní závěry by bylo potřeba mnohem více experimentů na větších datech, pro které se standardní metody akcelerují například pomocí grafických koprocesorů *CUDA*.

Zajímavým jevem je porovnání výsledků u testovacích a trénovacích dat maligní formy rakoviny, kdy u vyššího procenta mutace vzrůstá chyba typu II, stejně tak jako rozdíl správně klasifikovaných vzorků u trénovací a testovací množiny. Dodejme, že statistická data vznikají vždy z datového setu A.

V tabulkách 6.1.3 a B.1 můžeme vidět průměrné a maximální hodnoty pro přesnosti měření v bodě α (zvýrazněny jsou nejvyšší hodnoty na testovací množině pro zvolený parametr n_c). Vyplývají z nich zásadní skutečnosti týkající se tvaru topologie při vlivu na robustnost sítě. Nepodařilo se prokázat významnost parametru n_n. Z průměrných a maximálních hodnot se ukazuje, že přesnosti na trénovací množině jsou u všech zkoumaných genotypů srovnatelné. Liší ze přesnosti na množině testovací. Zejména u maximálních naměřených přesností na testovací množině je markantně vidět rozdíl v přesnostech dosažených malými genotypy ($n_c = 50$) oproti genotypům větším. Srovnejme například maximální přesnosti dosažené na testovací množině genotypem $n_c = 50$, s 10% mutací a strategií $ES(5+5)$ oproti genotypu s $n_c = 200$.

Nejlepších hodnot z hlediska bodu α jsme dosahovali s genotypy $n_c = 50$, vyšší míra mutací přitom nestimulovala populaci k lepším výsledkům. Pokud srovnáme průměrné hodnoty přesností na testovacích množinách v bodě α pro stejné konfigurace, lišící se jen mírou mutace z tabulky B.1, vyplyne pozitivní vliv nižších mutací.

n_c	n_n	10% mutace		20% mutace	
		$ES(1+9)$	$ES(5+5)$	$ES(1+9)$	$ES(5+5)$
50	10	99.00% ; 95.50% (5650)	99.00% ; 98.50% (1464)	98.50% ; 95.00% (3553)	97.50% ; 96.00% (2639)
	25	97.50% ; 95.50% (54)	97.00% ; 96.00% (539)	98.00% ; 94.50% (1369)	98.50% ; 94.50% (2851)
	40	98.50% ; 95.00% (5683)	98.00% ; 97.00% (2038)	98.00% ; 96.00% (1633)	98.00% ; 95.00% (6908)
100	10	98.50% ; 95.50% (3488)	98.50% ; 95.50% (5060)	98.00% ; 95.50% (9582)	98.50% ; 95.00% (7806)
	25	98.50% ; 95.00% (1288)	98.50% ; 95.50% (10483)	98.00% ; 95.00% (2844)	98.00% ; 95.50% (10632)
	40	97.50% ; 95.00% (1634)	98.00% ; 95.00% (998)	97.50% ; 95.50% (1227)	98.00% ; 95.50% (5649)
200	10	97.00% ; 95.00% (919)	99.00% ; 95.50% (2999)	98.50% ; 95.00% (9137)	97.50% ; 95.50% (2057)
	25	98.50% ; 96.00% (3847)	95.00% ; 96.00% (237)	98.50% ; 95.50% (1221)	98.00% ; 96.50% (1559)
	40	97.50% ; 95.00% (424)	98.50% ; 95.00% (7568)	96.50% ; 95.00% (150)	97.50% ; 95.50% (3464)
300	10	98.50% ; 96.00% (2292)	98.50% ; 95.50% (10756)	98.50% ; 95.00% (949)	97.50% ; 95.50% (626)
	25	98.00% ; 94.50% (107)	98.50% ; 96.00% (4945)	98.50% ; 93.50% (10624)	98.00% ; 95.00% (6706)
	40	97.50% ; 94.50% (2016)	98.00% ; 95.50% (6434)	98.00% ; 94.50% (1226)	97.50% ; 95.50% (2459)

Tabulka 6.1: *Maximální hodnoty přesností klasifikace na trénovací a testovací množině s příslušným počtem generací k jejímu dosažení v bodě α. Formát buňky tabulky je: přesnost na trénovací množině ; přesnost na testovací množině (počet generací). Modře jsou znázorněny nejlepší výsledky pro konfiguraci definovanou mírou mutace, evoluční strategií a n_c.*

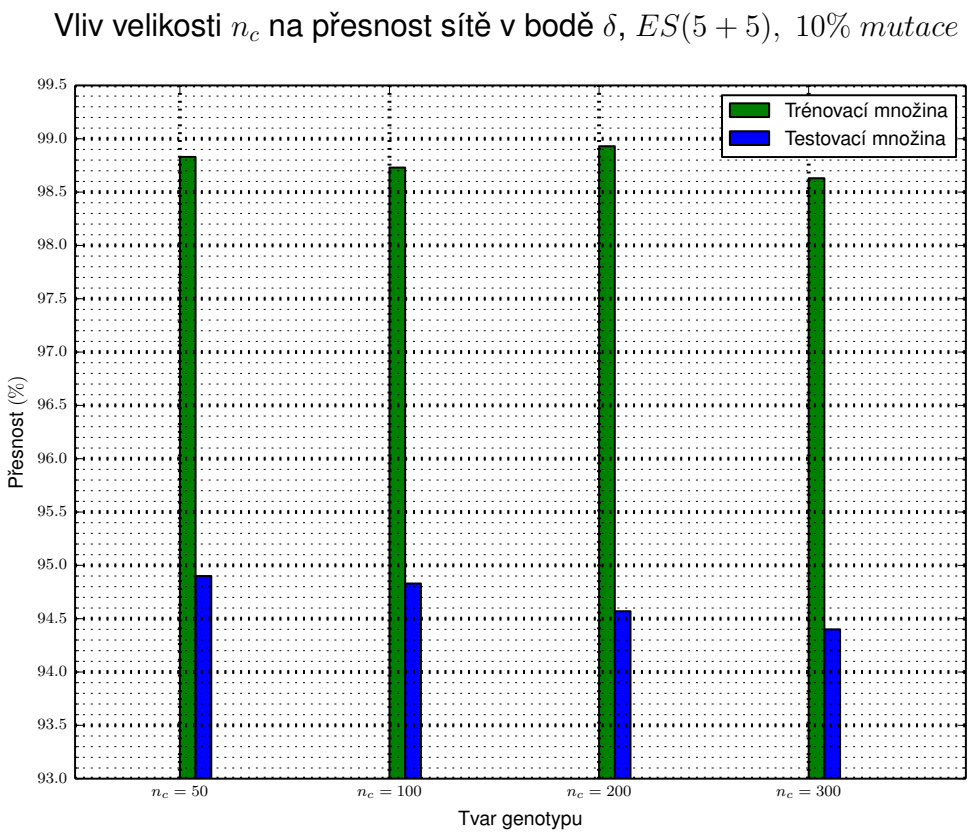
Statistické data z tabulek B.3 a B.4, vztahujícím se k bodu β měření, potvrzují trend nastíněný v bodě α. U větších genotypů a vyšší míry mutace jsou průměrné a absolutní hodnoty přesnosti na testovací množině nižší než u menších genotypů či nižší míry mutace.

Tabulky 6.2 a B.5 se vztahují k výsledkům po 11000 iteracích výpočtu definovaným bodem δ.

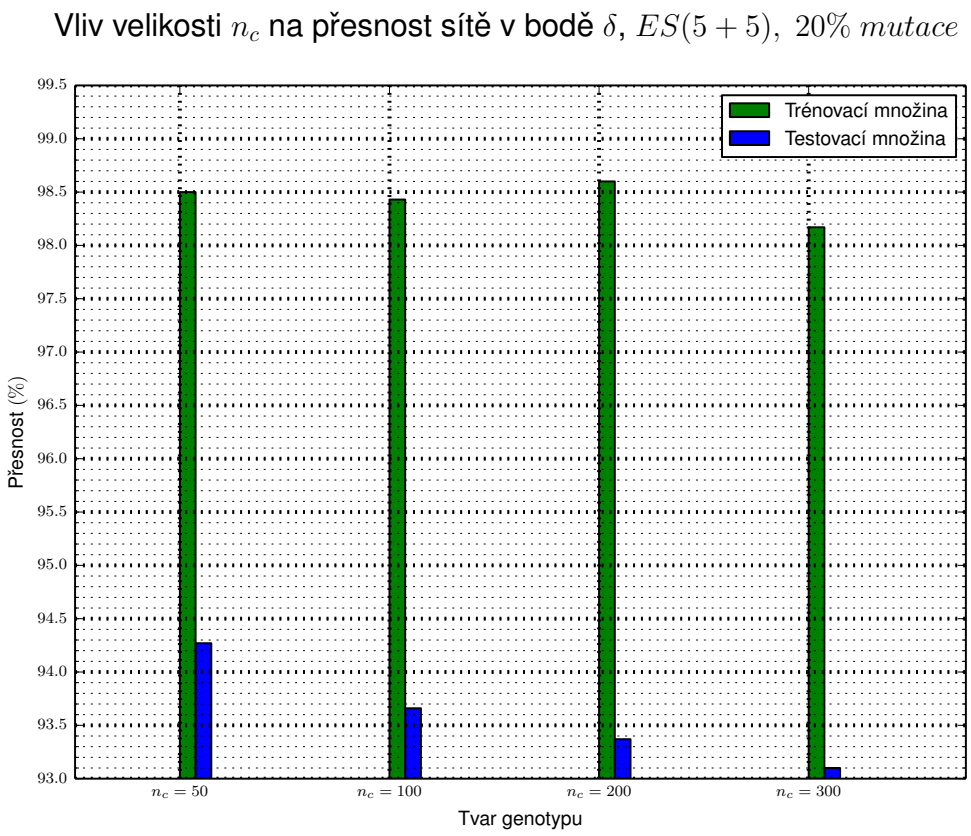
n_c	n_n	10% mutace		20% mutace	
		$ES(1+9)$	$ES(5+5)$	$ES(1+9)$	$ES(5+5)$
50	10	98.90% ; 94.30%	98.60% ; 94.30%	98.10% ; 92.00%	98.50% ; 94.60%
	25	98.90% ; 93.20%	99.00% ; 95.00%	98.20% ; 92.40%	98.50% ; 94.30%
	40	98.50% ; 93.40%	98.90% ; 95.40%	98.50% ; 91.70%	98.50% ; 93.90%
100	10	98.90% ; 93.50%	98.80% ; 95.00%	98.10% ; 94.90%	98.40% ; 94.80%
	25	98.50% ; 93.00%	98.90% ; 95.00%	98.40% ; 93.80%	98.50% ; 93.10%
	40	98.50% ; 92.80%	98.50% ; 94.50%	98.50% ; 93.00%	98.40% ; 93.10%
200	10	98.90% ; 94.10%	99.00% ; 95.30%	98.40% ; 93.40%	98.50% ; 93.50%
	25	98.80% ; 93.30%	98.90% ; 94.70%	98.50% ; 93.20%	98.90% ; 93.70%
	40	98.60% ; 91.80%	98.90% ; 93.70%	98.40% ; 93.20%	98.40% ; 92.90%
300	10	98.50% ; 92.70%	98.90% ; 94.00%	98.50% ; 94.40%	98.10% ; 94.20%
	25	98.10% ; 92.50%	98.60% ; 94.90%	98.40% ; 93.50%	98.40% ; 91.30%
	40	98.60% ; 92.70%	98.40% ; 94.30%	98.50% ; 93.40%	98.00% ; 93.80%

Tabulka 6.2: *Průměrné hodnoty přesností klasifikace na trénovací a testovací množině s příslušným počtem generací k jejímu dosažení v bodě δ. Formát buňky tabulky je: přesnost na trénovací množině ; přesnost na testovací množině. Modře jsou znázorněny nejlepší výsledky pro konfiguraci definovanou mírou mutace, evoluční strategií a n_c.*

K tabulce 6.2 se vztahují grafy (obrázky) 6.6 a 6.7. Ty jednoznačně potvrzují trend popsaný v předchozích dvou bodech z hlediska testovací množiny. U větších chromozomů se, oproti těm s nízkým n_c, objevuje při řešení více intronů (do fenotypu také přibude více uzlů, procentuální nárůst jejich počtu je ovšem mnohem menší nežli procentuální nárůst genotypu samotného), řešení se na trénovací množině vlivem většího počtu uzlů nijak dramaticky nezpřesňuje, naopak dojde k snížení přesnosti na testovací množině. Při srovnání obou grafů je také vidět, že výsledky získané vyšší mírou mutace obvykle uvíznou v lokálním, nikoliv globálním, maximu přesnosti.

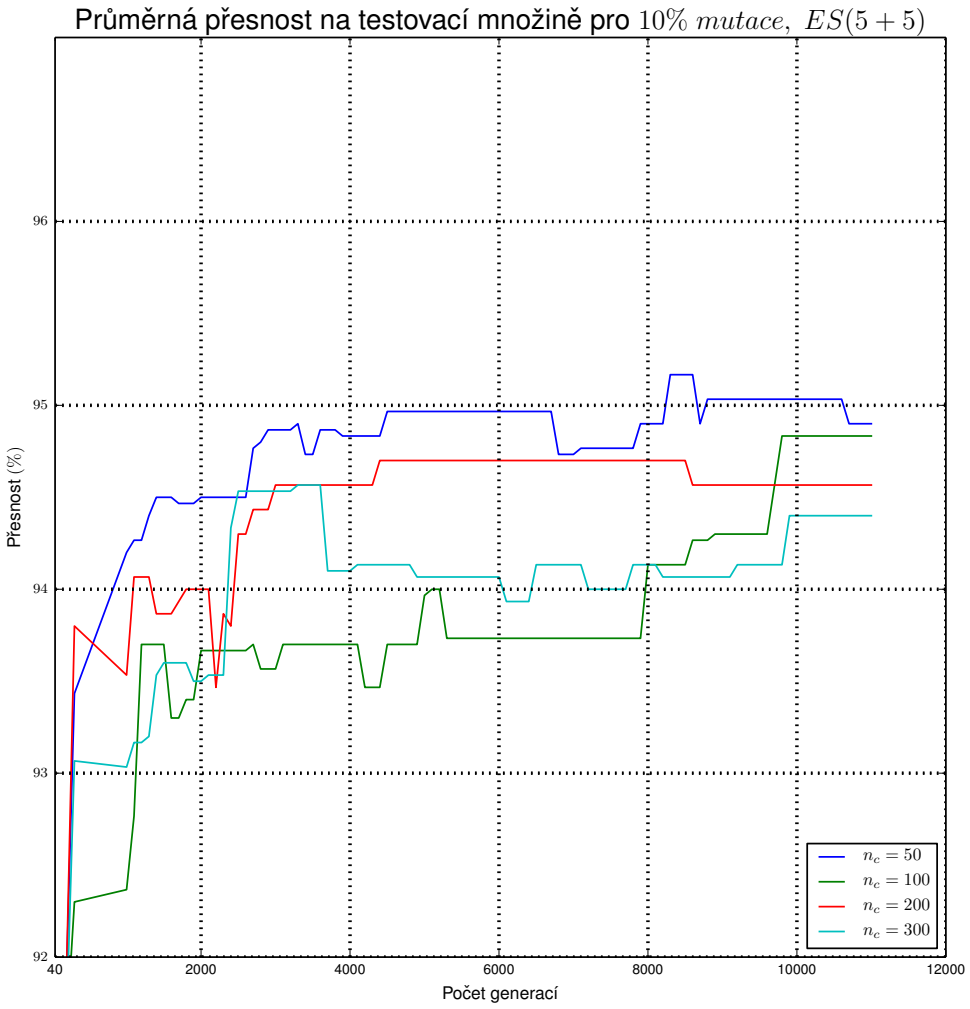


Obrázek 6.4: *Průměrná hodnota přeučení pro rozdílné velikosti chromozomu s nízkou mírou mutace.*

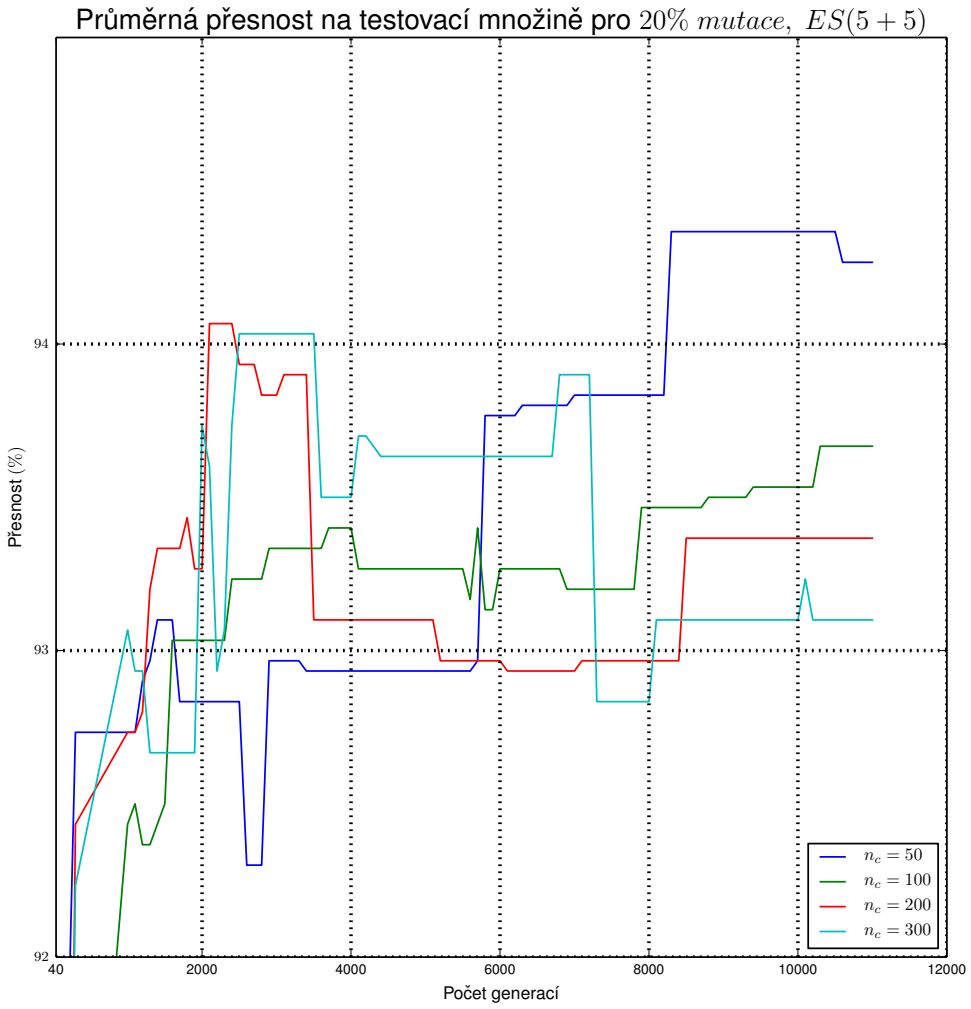


Obrázek 6.5: *Průměrná hodnota přeučení pro rozdílné velikosti chromozomu s vysokou mírou mutace.*

V grafech 6.6 a 6.7, které reprezentují průměry z maximálních přesností v dané generaci pro testovací množinu u detekce rakovinných buněk, je patrný negativní vliv vysokých mutací na hledání kvalitního řešení. Zdůrazněme, že mutace představuje ve své podstatě nesystematickou metodu evoluce, kdy velkými změnami v genotypu můžeme průměrnou kvalitu populace snížit. Čím vyšší míru této nesystematičnosti budeme do řešení zanášet, tím s větší pravděpodobností budeme svědky skokovitého chování v grafech průběhu tak, jak je tomu vidět například v grafu 6.7. Na rozdíl od grafu 6.6, v 6.7 konverguje bez větších propadů v přesnosti pouze chromozom s $n_c = 50$, kde byl poměr počtu mutovaných a aktivně zapojených uzlů ve výpočtu nastaven nejoptimálněji. U větších chromozomů jen mírně vzroste počet aktivně zapojených uzlů ve fenotypu. Oproti tomu v porovnání s menšími chromozomy (z hlediska n_c) rapidně vzrostla velikost intronů *(viz. definice 2.12)* a také počet mutovaných uzlů. Proto se i častěji objevovaly velké rozdíly v přesnostech na testovací množině během vývoje řešení. Celkově vysoká mutace vede k pomalejší evoluci z hlediska přesnosti vyhodnocení na testovací množině, což data potvrzují. Tím ovšem nevylučujeme, že vysoká mutace může u méně komplexních problémů nalézt požadované řešení rychleji než nízká míra mutace (právě díky většímu rozrůznění výsledků nad malým stavovým prostorem). Nízký počet kroků evoluce ovšem způsobuje problémy, které popíšeme v kapitole 6.2.



Obrázek 6.6: *Průběh průměrných hodnot na testovací množině při nízkých mutacích.*



Obrázek 6.7: *Průběh průměrných hodnot na testovací množině při nízkých mutacích.*

V tabulce 6.3 jsou shrnuty absolutní nejlepší výsledky měření. Jsou zde uvedeny i nejlepší výsledky pro sadu B, kterou jsme více statisticky nerozebírali, jelikož na ní kvůli časové náročnosti nebyl proveden dostatek výpočtů. Provedeny byly také experimenty s metodou subpopulací popsanou v kapitole 4.2.5. V tomto případě byla použita populace s celkem 20 jedinci s maximální počtem 11000 generací. Vytvářeli jsme 4 subpopulace s $ES(2+3)$ a rozrůzněním $n_c \in \{40, .., 60\}$ a $n_n = 10$. Tento postup neukázal žádné signifikantní zlepšení výsledků.

Hlavním pozitivem veškerých měření byl fakt, že ve všech případech z tabulky 6.3 jsme dosahovali 100% přesnosti na maligních datech. Nejlepších absolutních výsledků pro sadu A jsme dosahovali s genotypem $n_c = 50$ a to 99.5% na trénovací a 98.5% na testovací množině (výsledky ještě menších genotypů, např $n_c = 5$ byly srovnatelné s $n_c = 50$). Procentuální přesnosti nejlepších řešení jsou srovnatelná s výsledky z [1], kde maximální přesnost dosažená na 200 vzorkové trénovací množině za použití evoluční strategie $ES(1+9)$ s 10% mutací byla 99.5% a na testovací 98%.

Řešení z [1] vzniklo až po 100000 generacích a autor zde nepozoroval žádné známky přetrénování. Rozdílné jsou i topologie nejlepších řešení. Zatím co námi ověřené výsledky byly nejlepší pro nejmenší genotypy z hlediska počtu komponent (uzlů), autoři v [1] dosahovali nejlepších výsledků pro $n_c = 300,\ n_c = 200$ a $n_n = 25$.

Použitá sada	Rozrůznění populace	n_c	n_n	Trénovací množina			Testovací množina		
				Maligní data	Benigní data	Celková data	Maligní data	Benigní data	Celková data
Sada A	$ES(5+5)$,10% mutace	50	10	100.00% 100/100	98.00% 98/100	99.00% 198/200	100.00% 100/100	97.00% 97/100	98.50% 197/200
Sada A	subpopulace,10% mutace	40 – 60	10	100.00% 100/100	99.00% 99/100	99.50% 199/200	100.00% 100/100	97.00% 97/100	98.50% 197/200
Sada B	$ES(1+9)$,10% mutace	50	40	99.17% 120/121	98.19% 217/221	98.54% 337/342	100.00% 118/118	97.31% 217/223	98.24% 335/341
Sada B	subpopulace,10% mutace	50	40	98.35% 119/121	97.29% 215/221	97.66% 334/342	100.00% 118/118	96.86% 216/223	97.95% 334/341

Tabulka 6.3: *Maximální absolutní naměřené výsledky přesnosti. Každá buňka obsahuje procentuelní přesnost na vybrané množině a také absolutní počet správně klasifikovaných vzorků z množiny dané velikosti*

V [1] bylo provedeno také statistické měření pomocí křížové validace. Nejprve byla celá datová množina rozdělena na 10 přibližně stejných dílů. Jejich kombinacemi vzniklo 10 datových setů, každý z nich obsahoval 9 trénovacích bloků a jeden blok testovací. Pro každý dataset bylo provedeno 10 měření, celkové výsledky byly průměrovány. Přesnost dosažená v [1] potom byla 99.02% na trénovací a 97.99% na testovací množině. Tento výsledek předčil většinu porovnávaných metod. Naše výsledky byly získány jiným statistickým způsobem, proto nejsou zcela srovnatelné. Pro bod α byla nejlepší průměrná přesnost s množinou A a nastavením $n_c = 50$, $ES(5+5)$ a 10% mutací na trénovací množině 98.40%, resp. 96.20% na testovací množině. Průměrná přesnost na testovací množině datasetu B, obsahujícího celkem 682 záznamů, byla pro bod α (se stejnou konfigurací $n_c = 50$, $ES(5+5)$ a 10% mutací) 98.10% na testovací množině, čímž se koevoluční metoda může srovnávat podle [1][7] s nejlepšími dosaženými výsledky dalšími metodami (stručný souhrn v tabulce 6.4). Ve srovnání s [1] jsme se více zaměřili na zachování robustnosti sítě kontrolou bodu časného zastavení.

Metoda	Přesnost (%)
Koevoluce CGP a neuronových sítí	98.10
EPNet	97.25
MPANN	98.10
BP A	98.10
RBF	89.96
LVQ	46.10
CL	64.31
PNN	97.77
NegBoost s $\lambda = 0$	98.30

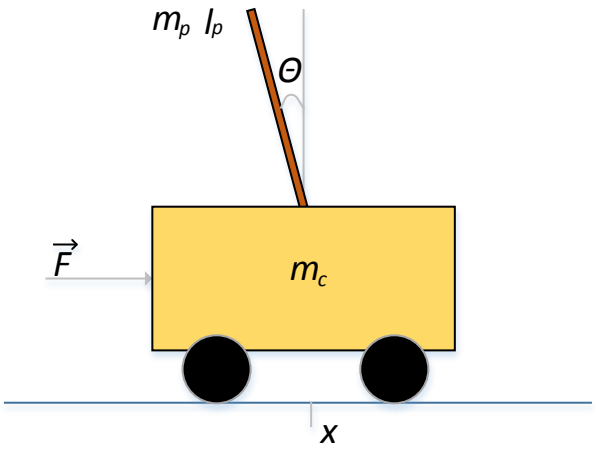
Tabulka 6.4: *Průměrné výsledky na testovací množině pro různé metody uvedené v [1]. Autoři zde přesně neuvádějí detaily nastavení metod při učení. Vždy je však použito množin vytvořených z Wisconsinského datového setu. Berme proto hodnoty v tabulce jako orientační. My uvádíme průměrnou přesnost sítě nad datasetem B s nastavením* $n_c = 50$, $ES(5+5)$ *a* 10% *mutací.*

6.2 Problém vyvažování tyče

Jedná se o základní benchmarkovou úlohu, použitelnou v mnoha obměnách.

6.2.1 Popis modelu a jeho použití

Mějme fyzikální model, jehož schéma je na obrázku 6.8. Vozík o hmotnosti m_c se pohybuje po rovině, pozice jeho těžiště vůči podkladu určuje souřadnice x. Na vozíku je postavena na počátku každého měření ve vratké rovnovážné vertikální poloze tyč. Její délku definujeme jako l_p a hmotnost m_p. Úhel θ svírá tyč se svislou rovinou po svém vychýlení.



Obrázek 6.8: *Vyvažovací model.*

Cílem je pohybovat vozíkem za pomoci konstantní síly $\vec{F}$ působící na jeho těžiště směrem vlevo nebo vpravo tak, aby byly zachovány dvě podmínky. Pozice těžiště vozíku zůstane v intervalu $I_x =< x_{min}; x_{max} >$ a tyč se nevychýlí z interval $I_\theta =< \theta_{min}; \theta_{max} >$

Definujme rovnice fyzikálního systému, které popisují jeho dynamiku v případě působení síly $\overrightarrow{F}$ [2]. V tabulce 6.5 je vysvětlen význam použitých proměnných. Jsou zde uvedeny také použité implicitní hodnoty konstant shodné s [7].

v_c	Rychlost vozíku	Inicializováno na $0ms^{-1}$
a_c	Zrychlení vozíku	Inicializováno na $0ms^{-2}$
x_c	Pozice těžiště vozíku	Inicializováno na $0m$
θ_p	Úhel vychýlení tyče	Inicializováno na $0rad$
ω_p	Úhlová rychlost tyče	Inicializováno na $0rads^{-1}$
ε_p	Úhlové zrychlení tyče	Inicializováno na $0rads^{-2}$
g	Tíhové zrychlení	$9.81ms^{-2}$
$\overrightarrow{F}$	Konstantní vektor síly působení	$10N$
τ	Perioda působení síly F	$0.02s$
m_c	Hmotnost vozíku	$1kg$
m_p	Hmotnost tyče	$10kg$
l_p	Délka tyče	$0.5m$
x_{min}	Minimální souřadnice těžiště	$-2.4m$
x_{max}	Maximální souřadnice těžiště	$2.4m$
θ_{min}	Maximální výchylka tyče (-)	$0.4094rad$, pro dopřednou síť $0.2094rad$
θ_{max}	Maximální výchylka tyče (+)	$0.4094rad$, pro dopřednou síť $0.2094rad$

Tabulka 6.5: *Proměnné systému*

Systém rovnic 6.1 popisuje aktualizované hodnoty vztahující se k poloze, rychlosti a zrychlení vozíku, stejně jako k úhlu sevřeného mezi svislou osou a tyčí, úhlové rychlosti a zrychlení tyče vzhledem k ose otáčení při působení síly $\overrightarrow{F}$ zvoleným směrem.

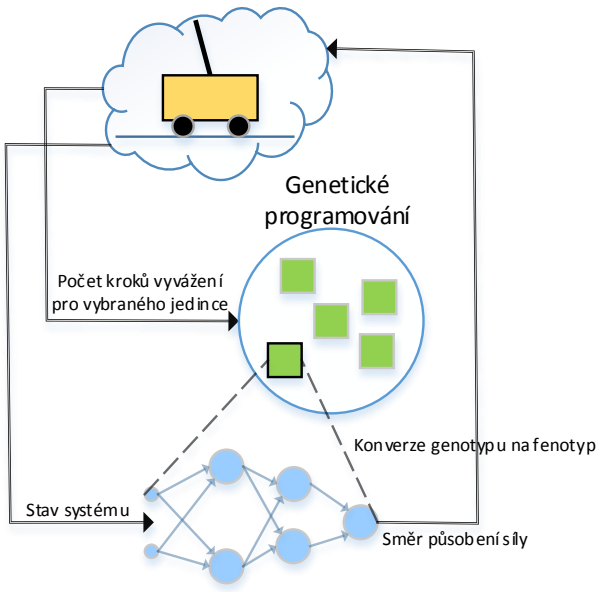
$$
\begin{aligned}
x_c &= x_c + \tau v_c \\
\theta_p &= \theta_p + \tau \omega_p \\
\varepsilon_p &= \frac{g\ sin(\theta_p) + cos(\theta_p)\frac{-F - m_p l_p \omega_p^2 sin(\theta_p)}{m_c + m_p}}{l_p(\frac{4}{3} - \frac{m_p cos^2(\theta_p)}{m_c + m_p})} \\
\omega_p &= \omega_p + \tau \varepsilon_p \\
a_c &= \frac{F + m_p l_p(\omega_p^2\ sin(\theta_p) - \varepsilon_p cos(\theta_p))}{m_c + m_p} \\
v_c &= v_c + \tau a_c
\end{aligned}
\tag{6.1}
$$

Cílem evolvované sítě bude vyvažovat tyč v uvedených mezích. Ohodnocení jedince modelem začíná s tyčí, která je ve svislé poloze v klidu, poloha těžiště vozíku má x souřadnici $x = 0m$. Takto definujeme výchozí stav modelu pro každý experiment. Síť reaguje na stav modelu, který své stavové proměnné přináší na vstup sítě.

Počet vstupů je odlišný pro rekurentní a dopřednou alternativu evolvované neuronové sítě. Dopředná síť má celkem 4 vstupy pro proměnné v_c, x_c, θ_p a ω_p. Rekurentní alternativa má za vstup pouze x_c a θ_p. Zpětnovazebná spojení by měla poskytnout dostatečnou aproximaci první a druhé derivace polohy vozíku vzhledem k času, pomocí níž získáme jeho rychlost a zrychlení a tak i ucelenější informaci, jakým způsobem působit na vozík silou.

Výstupní hodnota sítě (síť má pouze jeden výstup) je porovnána s hodnotu 0, je tedy důležité pouze znaménko výstupu, které rozhodne o směru působení konstantní síly $\overrightarrow{F}$. Tou bude tyč vyvažována. Tato interakce probíhá maximálně po určený počet cyklů N nebo do

chvíle, než parametry systému vozík - tyč nepřekročí limity I_θ nebo I_x. Každý jedinec takto komunikuje s výše definovaným systémem a na základě doby vyvažování tyče je ohodnocen. Model při každém ohodnocení sítě definuje na počátku ohodnocení $f_v = 0$. Při každém úspěšném kroku vyvažování hodnotu f_v inkrementujeme a zároveň je dekrementována N (o 1). Model pokračuje v ohodnocování sítě dokud platí podmínka $x \in I_x \wedge \theta \in I_\theta \wedge N > 0$. Po ohodnocení celé populace jedinců, kdy každý začíná vyvažování z popsané rovnovážné polohy, jsou vybráni nejlepší do další generace dle evoluční strategie (viz. obrázek 6.9).



Obrázek 6.9: *Schéma průběhu algoritmu. Detail zachycuje jak jedinec komunikuje se systémem do chvíle, než se tyč nevychýlí z limitních pozic, poté je ohodnocen. Evoluce je řízena algoritmem genetického programování. Ve schématu je kladen důraz na ukázku způsobu ohodnocení jedince.*

6.2.2 Konfigurace výpočtu

Při testování použijeme dva typy koevolučních návrhů. Přímé zakódování dopředné neuronové sítě a přímé zakódování rekurentní neuronové sítě vycházející z *Jordanovy neuronové sítě*. Obě sítě, rekurentní i dopřednou alternativu, budeme testovat pro různá nastavení topologie genotypu s populacemi o 10 jedincích. Použijeme přitom sadu aktivačních funkcí $y = tanh(x)$ a $y = \frac{1}{1+e^{-x}}$. Výstupní hodnota sítě bude upravena pomocí hyperbolického tangentu, jelikož chceme podpořit vliv znaménka výstupu sítě pro determinaci směru působení síly na vozík. Počet vstupů sítě bude u dopředné alternativy $n_i = 4$ s aritou uzlů $n_n = 4$ a u rekurentní $n_i = 2$ s $n_n = 3$. Obě sítě mají $n_o = 1$.

Definujme také maximální počet kroků vyvažování $N = 100000$, což je zároveň maximální ohodnocení f_v jedince. V experimentech také provedeme úpravy ohodnocení s cílem ovlivnit robustnost sítě.

Pro každou statistickou hodnotu proběhlo celkem 100 měření, jejichž výsledky byly průměrovány nebo mezi nimiž byla hledána nejlepší řešení.

6.2.3 Vyvažování dopřednou sítí

V případě evoluce dopředné neuronové sítě jsme provedli 3 základní typy experimentů. Jejich cílem je identifikovat nejlepší konfiguraci sítě a srovnat ji s výsledky z [7]. Další 2 sady experimentů se zabývají robustností sítě a možnostmi, jak ovlivnit pomocí nastavení ohodnocení její chování.

Pro dopřednou síť jsme zvolili interval povoleného vychýlení tyče při vyhodnocování $I_\theta =< -0.2094\ rad;\ 0.2094\ rad >$. Tento interval byl zvolen ze dvou důvodů. Evoluce dopředné sítě pro vyvažování je oproti úloze pro síť rekurentní jednodušší úlohou z hlediska výpočetní náročnosti. Dopředná síť implicitně pracuje s více informacemi o stavu tyče a vozíku. Proto se snažíme k limitům možností koevoluční techniky dostat pomocí zpřísnění omezujících podmínek modelu. Druhým důvodem je možnost srovnání s [7]. Proto jsme pro toto měření převzali veškeré hodnoty konstant.

V tabulce 6.6 uvádíme průměrný počet generací k dosažení maximálního počtu kroků ($N = 100000$) vyvažování tyče. Neuvádíme minimální počet generací, jelikož u všech uvedených nastavení se nám podařilo získat optimální neuronovou síť po vzniku druhé generace. V krajním případě to znamená, že jsme provedli 10% mutaci nejlepších jedinců z počáteční generace, ve které se nachází náhodně vytvořená populace. U $ES(5+5)$ vzniklo v druhé generaci 5 nových jedinců mutací 5 nejlepších rodičů. Celkem byla tedy potřeba 15 jedinců k dosažení vytyčeného cíle.

Průměrné hodnoty výsledků nasvědčují o srovnatelných vlastnostech popsaných v testech s buňkami rakoviny v 6.1.3. Z hlediska velikosti n_c nejrychleji konvergovaly k optimu chromozomy s nejmenší hodnotou $n_c = 5$. Nejlepšího řešení jsme dosáhli při 20% mutaci a $ES(1+9)$. Optimální počet vyvažovacích kroků dosahovala síť vyevolvovaná v průměru v 23.generaci. Naše závěry jsou v kontradikci s [7], kde pro stejná nastavení s $ES(1+9)$ dosahovali autoři v průměru nejlepších řešení u $n_c = 15$ a to 21 generací při 10% mutaci. Se stejnou konfigurací jsme dosahovali průměru 49 generací pro nalezení optima a celkově byla tato konfigurace až 6 nejlepší z 12 zkoumaných nastavení.

Při srovnání vlivu mutací bez přihlédnutí k vedlejším vlastnostem sítě se ukázalo, že výsledky stejné konfigurace s rozdílným procentem mutace vždy vyzněly lépe pro větší procento mutace. Mutace tedy byla pozitivním faktorem při rychlosti hledání požadovaného řešení. Tento fakt potvrdil i výzkum z [7]. Ačkoliv vyšší mutace hledala řešení svou větší diverzifikací stavového prostoru rychleji, v testech s robustností neuspěla. Robustnost řešení více jak s mutací souvisí s počtem kroků provedených k získání řešení. Řešení nalezená nejmenším počtem kroků nebyla ta nejlepší z pohledu robustnosti, což bude v detailu popsáno v následujícím odstavci. Srovnání evolučních strategií z hlediska rychlosti generování požadovaného řešení není zcela relevantní, jelikož $ES(5+5)$ produkuje pro novou populaci maximálně 5 nově zmutovaných jedinců, zatímco $ES(1+9)$ 9. Proto stavový prostor zkoumá rychleji, ačkoliv může přicházet o potenciálně dobrá řešení, jelikož uvažuje pouze 1 rodiče, jakožto tvůrce nové populace.

$n_c \times n_r$	10% mutace		20% mutace	
	$ES(1+9)$	$ES(5+5)$	$ES(1+9)$	$ES(5+5)$
5×5	32.93	42.08	22.29	34.42
10×10	50.77	67.60	36.46	54.04
15×15	48.62	88.91	52.34	80.43

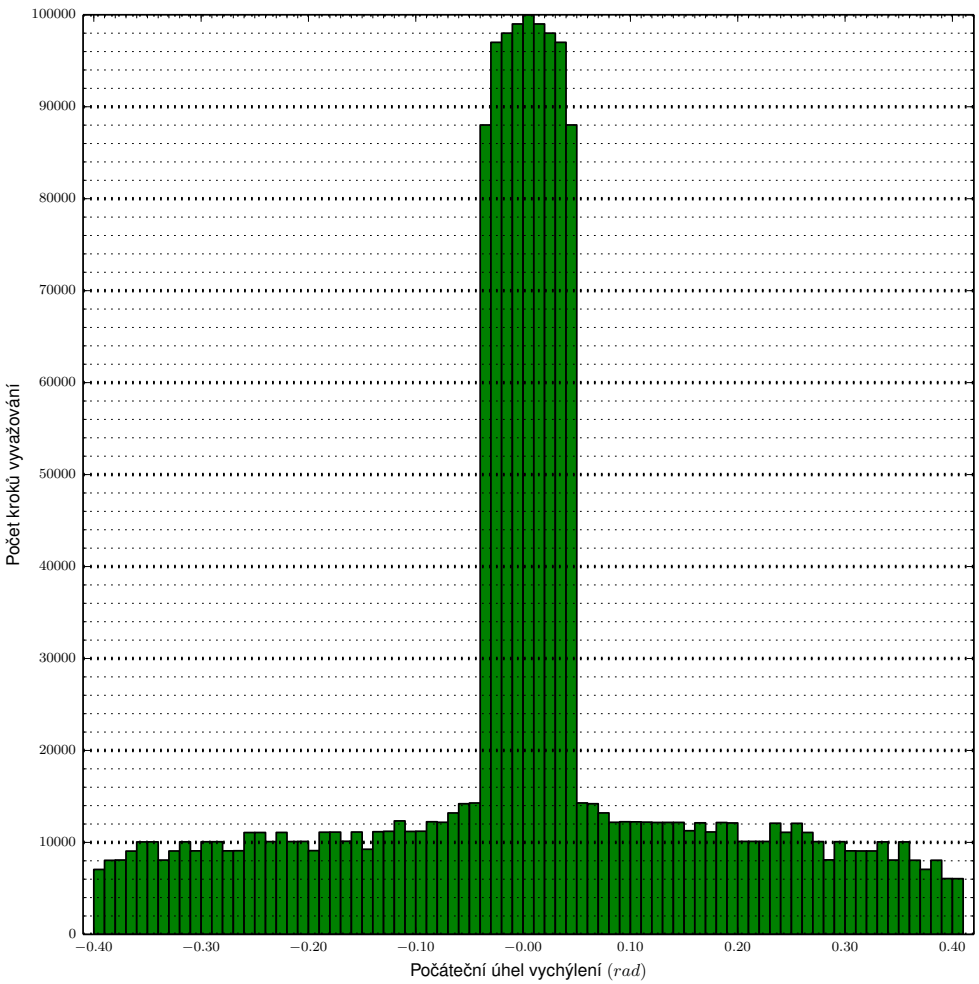
Tabulka 6.6: *Průměrný počet generací k dosažení vyvažování po dobu* $N = 100000$ *kroků. Ve všech* 1200 *provedených experimentech jsme dokázali během* 11000 *generací evolvovat síť, která dokázala balancovat tyč v uvedených limitech* 100000 *kroků v řadě. Modře jsou znázorněny nejlepší řešení napříč mutacemi a evolučními strategiemi.*

Pro evolvované sítě jsme experimentovali s jejich robustností. Robustnost chápeme jako schopnost evolvované sítě vypořádat se s jiným počátečním nastavením modelu, nežli s nastavením, se kterým byly původně evolvovány. Při evoluci sítě začínalo vyhodnocení každého jedince s tyčí ve svislé poloze a toleranční interval pro úhel vychýlení byl $I_\theta =< -0.2094\ rad;\ 0.2094\ rad >$. Uvažujme červeně zvýrazněné konfigurace v tabulce 6.6. Vždy pro všech 100 experimentálních běhů jsme vybrali z 5 nejlépe vyevolvovaných sítí tu, u které byla nejvyšší suma kroků vyvažování S. Předpokládejme, že $M(\theta)$ je popsaný model, jehož nastavení se změní pouze tak, že tyč je na začátku vyhodnocování již vychýlena o úhel θ. Dále mějme funkci $eval : \alpha \times M \to \mathbb{N}$, která určí počet vyvažovacích kroků dané evolvované sítě reprezentované jedincem α nad modelem M, než je porušena jedna z omezujících podmínek. Pro model M_{robust} upravme omezující podmínku I_θ na $I_\theta =< -0.4094\ rad;\ 0.4094\ rad >$, krok pro sumu volme $0.1\ rad$ a uvažujme ohodnocení jedince α

$$S_\alpha = \sum_{\theta=-0.4\ rad}^{\theta=0.4\ rad} eval(\alpha, M_{robust}(\theta))$$

Pro takto vybrané sítě jsme připravili 3 histogramy C.1, C.2 a 6.10, představující vždy průměrný počet kroků vyvažování pro dané počáteční vychýlení tyče.

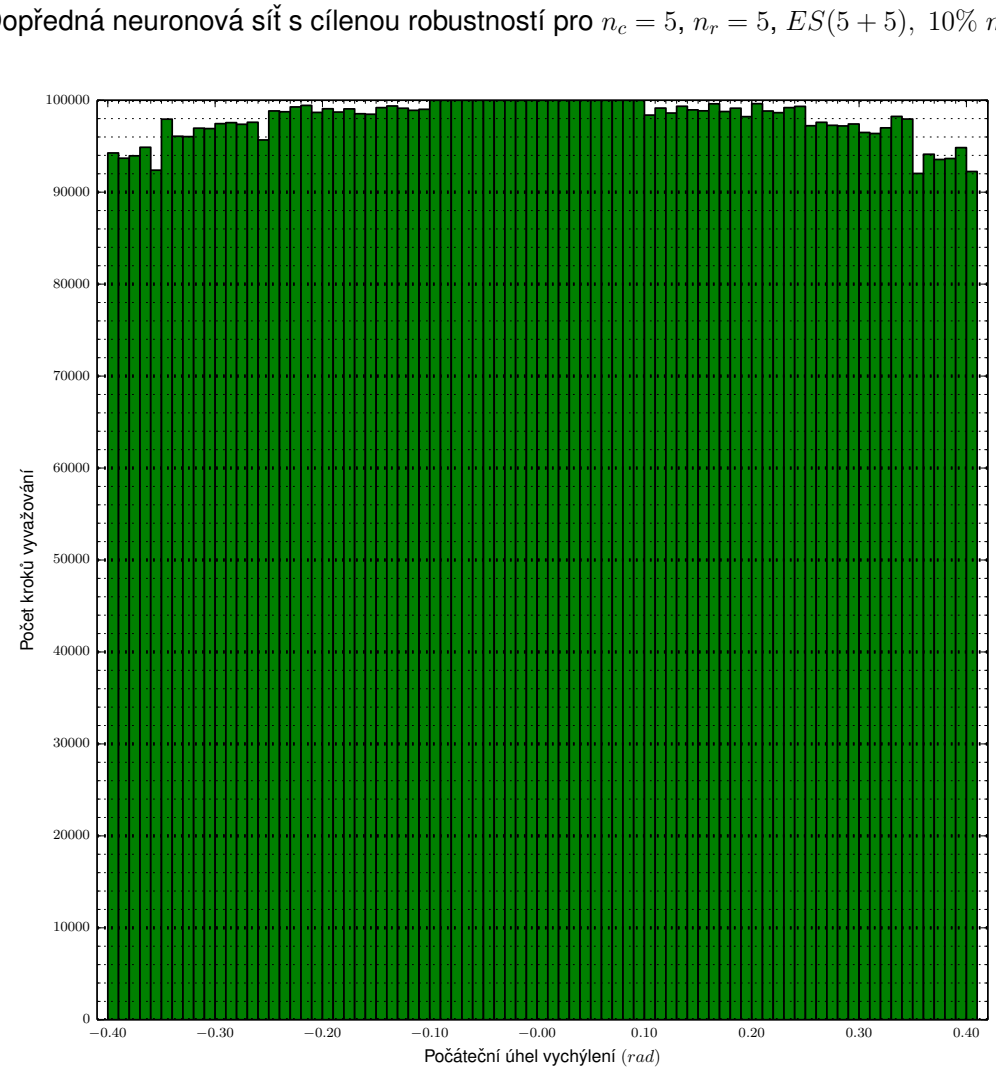
Z průměrných hodnot pro úhel $0\ rad$ ve zmíněných histogramech je zřejmé, že nejrobustnější řešení dle popsané metriky nebyla stejná řešení, která se objevila v tabulce 6.6, jelikož průměrné hodnoty v C.1 a C.2 nedosahují 100000. Přílišná specializace sítě nebyla připravena na velké úhly vychýlení, jelikož se s nimi reálně nikdy nepočítalo. Nejrobustnější řešení tak představovaly sítě, které k optimu konvergovaly největším počtem generací (tento fakt byl ověřen i v obecném měřítku). Řešení v podobě zakódovaní sítě se tak přirozeně postupně vyvíjelo, kdy předci nejlepšího řešení nedokázaly tyč dostatečně balancovat, přitom se síť učila vypořádat i s většími výchylkami. Tato schopnost se mohla přenášet generacemi dál, proto dokázaly jedinci poslední generace obstát nejlépe v testu robustnosti.



Obrázek 6.10: *Pro 100 testovacích běhů vygenerujeme vždy dopřednou neuronovou síť vyvažující tyč na vozíku. Učení začíná při svislé poloze tyče, snažíme se dosáhnout* 100000 *kroků vyvažování. Graf zobrazuje průměrný počet kroků vyvažování tyče, jestliže natrénovaná síť započne vyvažování s již vychýlenou tyčí.*

Z testu robustnosti vyplývala jednoznačná otázka. Dokážeme jenom za pomoci změněného způsobu ohodnocování evolvované neuronové sítě docílit toho, aby vzniklo robustní řešení. Pro tento experiment jsme si vybrali síť s parametrem $n_c = 5$. Model M_{robust} jsme nepoužili jen během závěrečného ohodnocování nejlepších jedinců, jako v předchozím experimentu. Model M_{robust} byl použit přímo během evolvování řešení. Při evoluci za použití modelu M_{robust} jsme definovali $f_v = S$ a dle tohoto ohodnocení vybírali jedince do další generace. Tím jsme metriku z předchozího experimentu přímo zakomponovali do evolučního procesu. Provedli jsme 100 statistických běhů algoritmu, vždy bylo použito všech 11000 generací k získání řešení. Nejlepší jedinec z poslední generace byl vybrán opět pomocí metriky S. Průměrné výsledky znázorňuje histogram 6.11. Histogram potvrzuje, že koevoluce

kartézského genetického programování a neuronových sítí představuje techniku, pomocí níž lze jen způsobem ohodnocení jedinců generovat velmi rozličné chování řešení. Konkrétní instanci vygenerované sítě můžeme vidět na obrázku C.6.

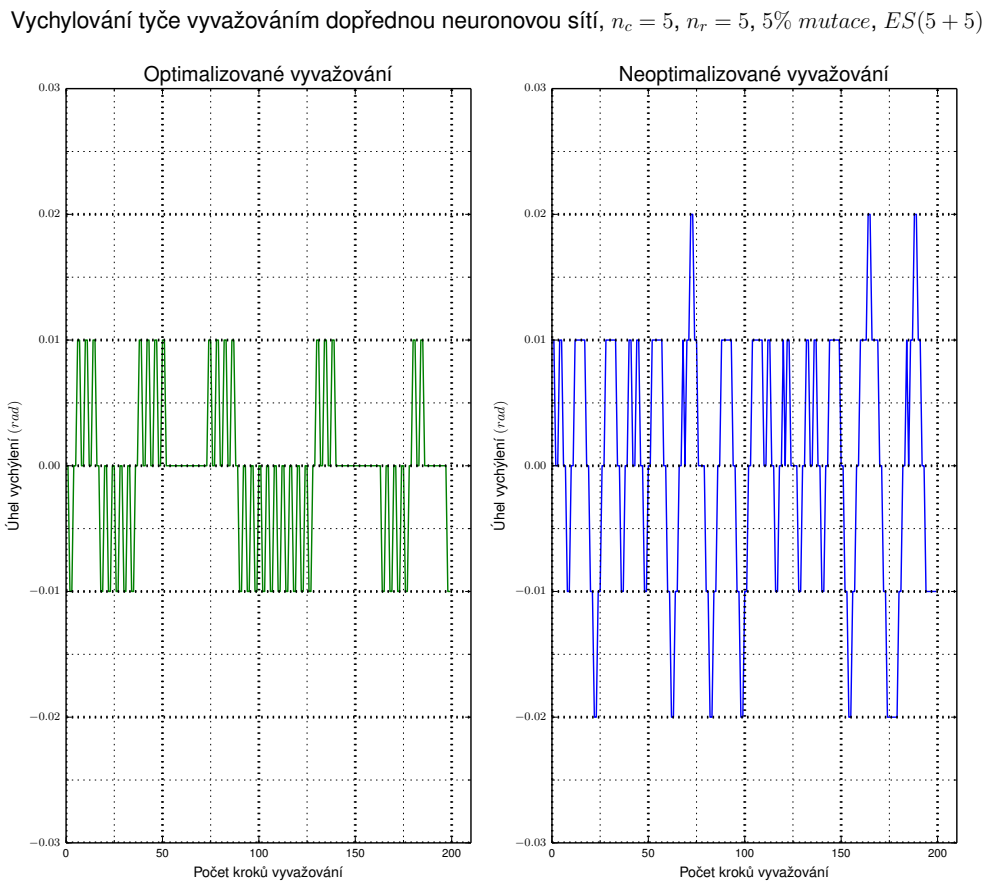


Obrázek 6.11: *Pro 100 testovacích běhů vygenerujeme dopřednou neuronovou síť vyvažující tyč na vozíku. Cíleně je generována robustní alternativa za pomocí metriky S. Graf zobrazuje průměrný počet kroků vyvažování tyče, jestliže natrénovaná síť započne vyvažování s již vychýlenou tyčí.*

Úspěšné experimenty s ovlivňováním vlastností řešení pomocí úpravy způsobu ohodnocování nás vedou k dalším pokusům. Pomocí úprav ohodnocení můžeme také ovlivnit to, jak moc se tyč během vyvažování vychyluje.

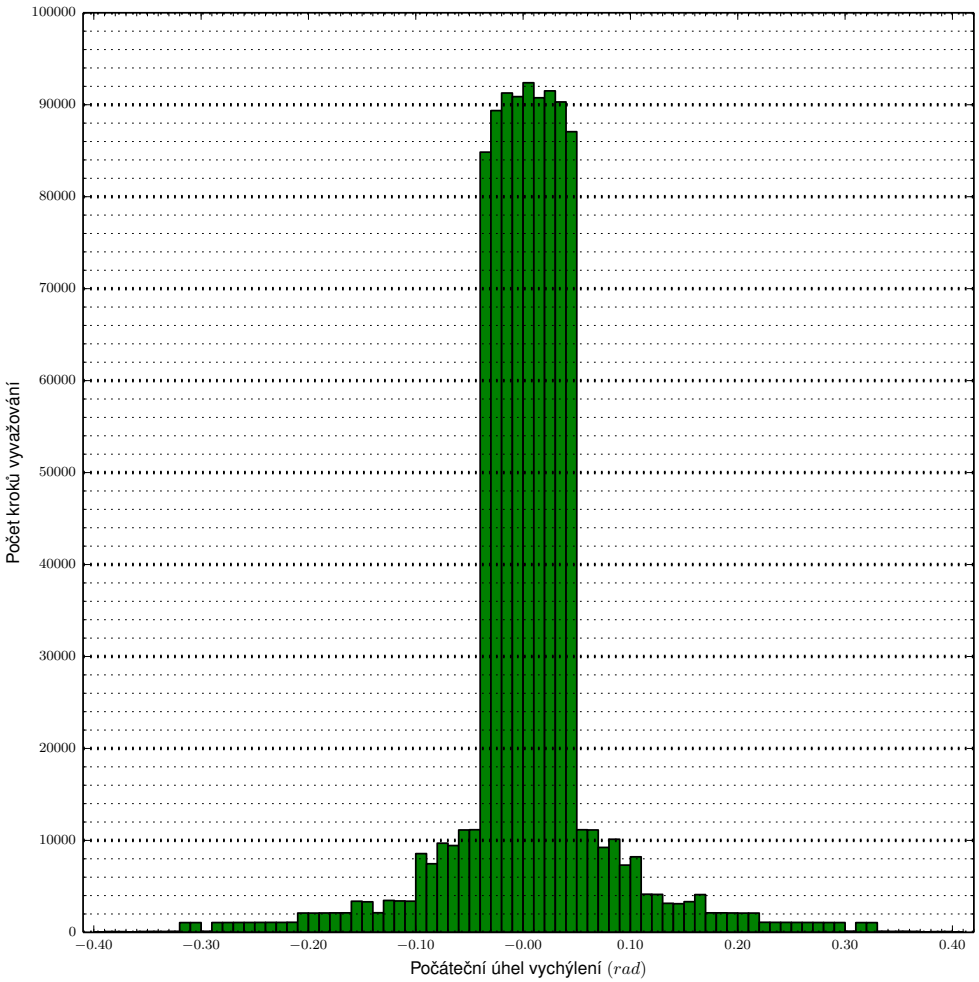
Mějme model s nastavením omezující podmínky pro vychylování $I_\theta =< -0.4094\,rad;\ 0.4094\,rad >$. Při vyhodnocování v klasickém modelu se při každém úspěšném kroku vyvažování inkrementuje hodnota f_v na počátku inicializovaná na $f_v = 0$. Upravené ohodnocování bude v každém úspěšném kroku inkrementovat hodnotu f_v, jenž zároveň sníží o velikost vy-

chýlení tyče v radiánech. Pokud jsou $f_v(t)$ a $f_v(t+1)$ průběžná ohodnocení jednoho fenotypu v po sobě následujících krocích, v klasickém modelu při použití funkce *eval* platí $f_v(t+1) = f_v(t) + 1$, v upraveném modelu $f_v(t+1) = f_v(t) + 1 - |\theta(t+1)|$. V grafu 6.12 je ukázán úsek kroků vyvažování dvou jedinců, jednoho evolvovaného pomocí optimalizovaného ohodnocování, druhého za pomoci neoptimalizované alternativy. V optimalizované alternativě se nám vychylování podařilo eliminovat.



Obrázek 6.12: *Optimalizovaná a neoptimalizovaná alternativa vyvažování. Graf znázorňující pohyb tyče při vyvažování shora.*

Pro 100 provedených experimentů jsme připravili histogram robustnosti takových řešení. Potvrdily se závěry o robustnosti řešení. Čím více je specializované na co nejmenší vychylování tyče při vyvažování, tím méně je schopno se vypořádat s abnormálními výchylkami, což můžeme vidět v grafu 6.13.



Obrázek 6.13: *Pro 100 testovacích běhů vygenerujeme vždy dopřednou neuronovou síť vyvažující tyč na vozíku se záměrnou optimalizací proti nadměrnému vychylování tyče. Učení začíná při svislé poloze tyče, snažíme se dosáhnout* 100000 *kroků vyvažování. Graf zobrazuje průměrný počet kroků vyvažování tyče, jestliže natrénovaná síť započne vyvažování s již vychýlenou tyčí.*

6.2.4 Vyvažování rekurentní sítí

V případě rekurentních sítí, na rozdíl od experimentů s dopřednou alternativou z kapitoly 6.2.3, jsme nedosahovali po 11000 generacích vždy optimálního řešení. Neuronová síť v tomto případě musí dopočítat parametry o zrychlení ze svých předešlých výstupů. Dopředná alternativa řešení dostává tuto informaci přímo z modelu. I z tohoto důvodu jsme nastavili omezující podmínky pro úhel vychýlení tyče na $I_\theta =< -0.4094\ rad;\ 0.4094\ rad >$.

V tabulce 6.7 jsou statisticky zpracované průměrné a maximální počty kroků vyvažování nejlepších řešení po 11000 generacích výpočtu (v tabulce 6.6 jsme pracovali s minimál-

ním počtem generací k dosažení 100000 kroků vyvažování, jedná se tedy o odlišné způsoby měření). Při porovnání velikostí parametru n_c se potvrzuje výhodnost menších chromozomů pro hledání optimálního řešení. Ze závěrů z kapitoly 6.2.3 vyplynulo, že rychleji byla požadovaná řešení hledána za pomocí vyšší mutace. V experimentech provedených s rekurentní neuronovou sítí byla doba evoluce pro všechny experimenty stejná, a to 11000 generací. A ačkoliv mutace hledaly rychleji optimální řešení u jednodušších problémů, experimenty s buňkami rakoviny v kapitole 6.1.3, stejně jako stávající experimenty s vyvažováním ukazují, že u složitějších problémů jsou řešení dosažená pomocí mutací za stejný počet generací horší z hlediska kvality, než řešení nalezená s nižší mírou mutace. Vysoká mutace představuje méně systematický přístup k prohledávání stavového prostoru, proto bez operátoru křížení nedosahuje lepších výsledků. Na obrázku C.7 můžeme vidět konkrétní instanci rekurentní sítě plnící úspěšně zadanou úlohu.

$n_c \times n_r$	10% mutace		20% mutace	
	$ES(1+9)$	$ES(5+5)$	$ES(1+9)$	$ES(5+5)$
5×5	21923.08 (100000)	13603.20 (100000)	15466.98 (100000)	12854.98 (95449)
10×10	17680.12 (100000)	15427.14 (100000)	14500.84 (100000)	12199.26 (25910)
15×15	20255.09 (100000)	14885.68 (100000)	15075.04 (100000)	11597.76 (23712)

Tabulka 6.7: *Průměrný a maximální počet vyvažovacích kroků po* 11000 *trénovacích cyklech. Maximum představuje* 100000 *kroků vyvažování. V závorkách uvádíme maximální dosaženou hodnotu pro síť vzniklou z chromozomu daných parametrů. Pro každou konfiguraci nastavení proběhlo* 100 *běhů. Modře jsou znázorněny nejlepší řešení napříč mutacemi a evolučními strategiemi.*

Pro testy robustnosti jsme si vybrali stejné konfigurace chromozomů (v tabulce označeny červeně) i stejný způsob výběru nejlepších jedinců pro statistické zpracování pomocí metriky S, jako u varianty s dopřednou neuronovou sítí. Z grafů C.4, C.3, C.5 vyplývá, že nejrobustnější řešení není reprezentováno jedinci, kteří dokáží vyvážit tyč s nejvyšším počtem kroků, jestliže vyvažování začne s počátečním úhlem $\theta = 0$ *rad*. Všimněme si, že v případě stejného počtu kroků evoluce jsou nejrobustnější (dle metriky S) řešení s nižší hodnotou n_c.

6.3 Problém pohybu v bludišti

Evolvovaná neuronová síť představuje agenta, který se pohybuje bludištěm reprezentovaném mřížkou na obrázcích 6.14 a 6.18. Agent plní jednotlivé cíle a snaží se přitom projít bludištěm s co nejmenším počtem kroků. Každý jedinec v každé generaci je ohodnocován nad modelem, ve kterém vychází z konstantní počáteční pozice a za modelem určený počet kroků musí splnit cíl. Model definuje také jeho ohodnocení, které je hlavním mechanizmem pro vynucení požadovaného chování agenta.

Pro problém bludiště uvedeme dva modely, řešené dvěma typy evolvovaných neuronových sítí. Pro přímé zakódovaní neuronové sítě použijeme model z kapitoly 6.4. Obecně poskytuje více informace o celkové poloze agenta a míře splnění jeho úkolu. V kapitole 6.5 popíšeme experimenty a výsledky nad modelem bludiště pro *Developmental Network*. Model klade na agenta větší nároky z hlediska zpracování signálů.

Agenti budou plnit tři základní úlohy. První z nich vyžaduje nalezení nejkratší cesty ze

startovní do cílové pozice *(kapitola 6.4.3)*. V druhé úloze bude agent hledat opět nejkratší cestu ze startovní pozice do cílové s tím rozdílem, že v nejkratší cestě je umístěna překážka, které se musí vyhnout *(kapitola 6.4.4)*. Poslední úloha je nejnáročnější. Agent se musí dostat do požadované pozice a odtud se vrátit zpět na pozici startovní *(kapitola 6.4.5)*. Pro všechny popsané úlohy byla provedena statistická měření, která budou dále diskutována.

6.4 Pohyb v bludišti pomocí rekurentní neuronové sítě

6.4.1 Popis modelu a jeho použití

Model vychází ze schematu na obrázku 6.14. Na počátku každého vyhodnocování aktuálně evolvovaného agenta je definována hodnota $E = 200$, obecně $E \in \{0 \dots 200\}$. Ta představuje zbývající počet kroků, který agent může v bludišti provést. Po každém kroku je hodnota E dekrementována (o 1). V případě, že $E = 0$ nebo je splněn cíl úlohy, vyhodnocování skončí.

Proměnná f_v reprezentuje celkové ohodnocení agenta. Jako nejlépe evolvovanou síť uvažujeme takovou, jejíž ohodnocení je nejvyšší. Ohodnocení agenta se postupně vytváří během jeho průchodu bludištěm. Na počátku je inicializováno vysokou hodnotou $f_i = 1000$. Celkové ohodnocení je potom suma dílčích ohodnocení agenta v jednotlivých krocích bludištěm a počáteční inicializace. Jelikož je vždy cílem nalézt nejkratší cestu ke splnění úlohy, v každém kroku je f_v dekrementována (o 1). Tím nepřímo zajistíme, že nejlépe ohodnocení jedinci plnili cíl pomocí nejméně kroků. V každé dílčí úloze je k základní popsané úpravě f_v přidána úprava nová, která je specifická pro danou úlohu.

1. `Nejkratší přímá cesta` - V této úloze zůstaneme u základního ohodnocení f_v, jelikož jediný cíl je projít bludiště nejkratší cestou.

2. `Nejkratší cesta s překážkou` - Pokud agent vstoupí jednou či vícekrát na pole, kde se nachází překážka, bude od jeho ohodnocení místo dekrementace f_v odečtena kladná konstanta $C = 200$. Pokud se agent dostane do cíle, je konstanta C místo dekrementace f_v k jeho ohodnocení přičtena. Nejhorší ohodnocení agenta proto bude $f_v = f_i - E - C + 1$. V případě, že délka optimální cesty k cíli je délky L, nejlepší ohodnocení agenta bude $f_v = f_i - L + 1 + C$. Pomocí správně zvolené konstanty C lze vhodně škálovat jedince podle chování v bludišti (dostal se do cíle, ale narazil na překážku; nedostal se do cíle a na překážku nenarazil atp..)

3. `Nejkratší uzavřená cesta` V případě nejkratší uzavřené cesty chceme pomocí ohodnocení nejprve oddělit ty agenty, kteří nalezli cíl a tyto dále evolvovat v následných generacích. K hodnotě f_v přičteme kladnou konstantu $C_a = 50$. Tím preferujeme agenty, kteří se dostanou k cílovému bodu. Ti agenti, kteří se dokáží vrátit zpět do počátku, jsou ohodnoceni vysokou hodnotou $C_b = 1000$. Tím jsou okamžitě vyčleněni z populace jako nejsilnější. Dále již probíhá jen optimalizace délky trasy.

Poslední sekci popisu modelu věnujme vstupním a výstupním hodnotám, pomocí nichž komunikuje model s agentem během vyhodnocování. Bludiště má 10 řádků a 10 sloupců indexovaných indexy z množiny $I = \{0 \dots 9\}$. Dva signály, které síť agenta zpracovává, jsou signály pozice agenta v bludišti. Ty jsou normalizovány do intervalu $I_n =< 0; 0.9 >$ tak, že platí

$$\forall i \in I \; \exists i_n : i_n \in I_n \wedge i_n = \frac{i}{10}$$

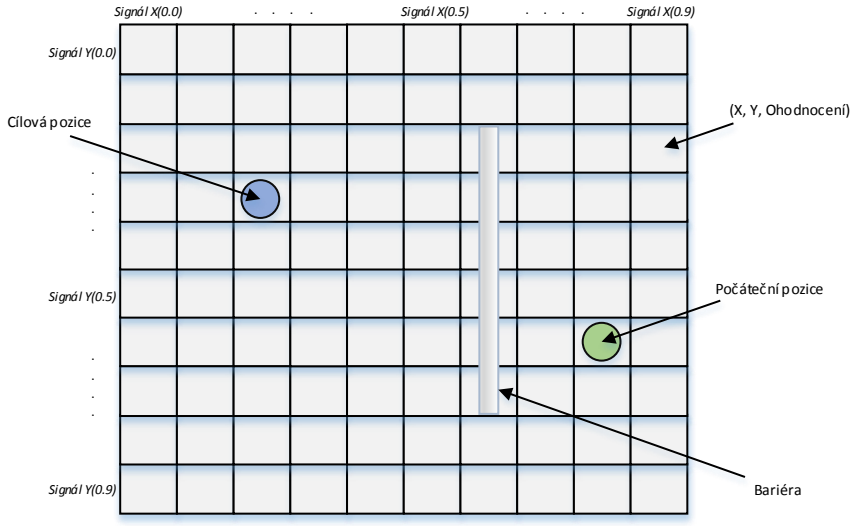
V případě úlohy, kde se agent vrací do počáteční pozice, přibude signál i_r určující, zda byla nalezena cílová pozice.

$$i_r = \begin{cases} 0 & \text{agent nenalezl cílovou pozici} \\ 1 & \text{agent již nalezl cílovou pozici} \end{cases}$$

Agent přivádí na vstup modelu vždy pouze jednu hodnotu o, $o \in < 0; 1 >$, která rozhodne o změně jeho polohy. Na každé pozici bludiště model určí, kolika směry se může agent vydat. Základní alternativa je sever, východ, jih a západ. V rozích a na okrajích bludiště se určitým směrem vydat nemůžeme. Například v levém horním rohu může agent postupovat na jih nebo východ. Počet možných alternativ v daném bodě označme A. Vždy jsou alternativy zvažovány v pořadí sever, východ, jih a západ. V případě, že dostupné alternativy budou indexovány indexem i od 0 v popsaném pořadí a funkce $tr : \mathbb{R} \to \mathbb{N}$ bude převádět reálné číslo na jeho celočíselný základ, pak index nově vybraného směru pohybu definujme jako

$$i = tr(Ao) \tag{6.2}$$

Neměnná počáteční a koncová pozice u všech experimentů je vybrána s tím cílem, aby síť musela simulovat komplexní chování, kdy se agent musí pohybovat na západ a sever, aby se z počáteční polohy dostal do polohy cílové. To znamená, že z hlediska definice výběru indexu i musí neuronová síť dokázat produkovat výstupní hodnoty z obou stran možného povoleného intervalu.



Obrázek 6.14: *Znázornění použité konfigurace bludiště s počáteční a cílovou pozicí, včetně umístění překážky. Každé pole je definováno hodnotou signálů určujících polohu a dodatečným ohodnocením, pomocí nějž je ovlivněna hodnota f_v a E při ohodnocení jedince.*

6.4.2 Konfigurace výpočtu

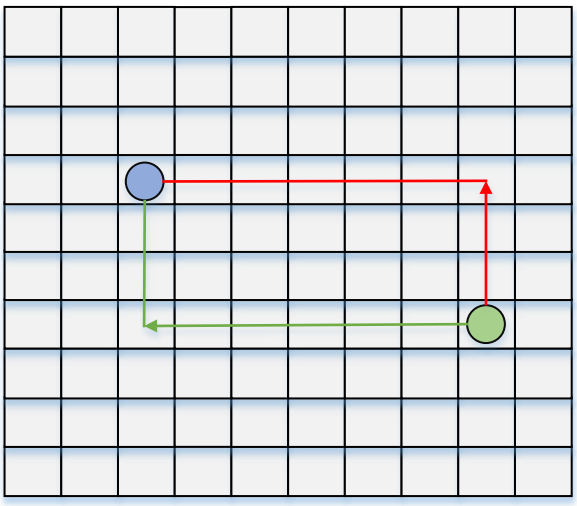
Populace 10 jedinců bude rozrůzněna pomocí 10% a 20% mutací a vybraných evolučních strategií. Jedinci jsou zakódované rekurentní neuronové sítě popsané v kapitole 4.2.3. Pro parametr n_c jsme vybrali nejúspěšnější konfigurace z předchozích experimentů s rakovinou a vyvažováním tyče. Množina aktivačních funkcí je tvořena $y = tanh(4x - 2)$ a $y = \frac{1}{1+e^{-4x}}$

(viz obrázek 4.8). Výstup celé sítě je upraven opět pomocí $y = \frac{1}{1+e^{-4x}}$, aby byl zaručen výstup sítě definovaný v definičním oboru modelu bludiště, $D(M) \in < 0; 1 >$. Rekurentní síť má v případě hledání uzavřené cesty $n_i = 3$. Kromě signálu pozice v bludišti model poskytuje boolovský signál i_r, který říká, zda agent nalezl cílovou pozici a měl by se vracet zpět do počáteční pozice. V ostatních případech je $n_i = 2$ a $n_o = 1$. Arita uzlů chromozomu reflektuje přítomnost zpětnovazebních uzlů pro každý výstup sítě, proto $n_n = n_i + n_o$.

Maximální počet generací je omezen horní hodnotou 11000.

6.4.3 Nejkratší přímá cesta

Agent měl za úkol nalézt nejkratší cestu z bodu A do bodu B. Tento úkol se ukázal jako triviální, jelikož každá instance z 1200 experimentů dokázala nalézt nejkratší cestu, v našem případě délky 9. Statistické hodnoty a tvar trasy můžeme vidět v tabulce 6.8 a na obrázku 6.15. Zvýšení míry mutace, stejně jako u vyvažování tyče s vozíkem, znamenalo rychlejší nalezení požadovaného řešení.



Obrázek 6.15: *Dvě nejobvyklejší nejkratší trasy, kterými agent našel cíl.*

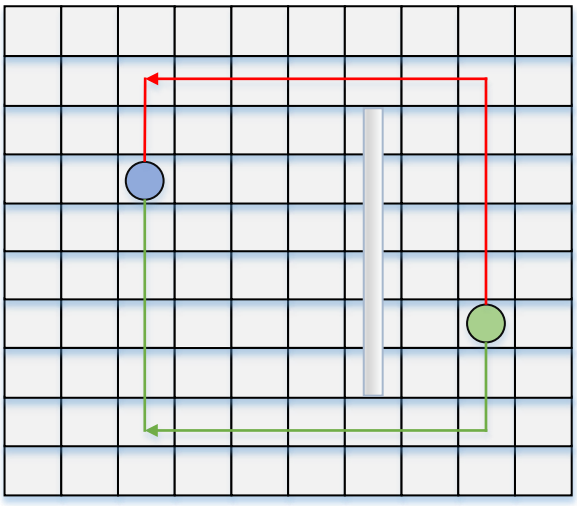
$n_c \times n_r$	10% mutace		20% mutace	
	$ES(1+9)$	$ES(5+5)$	$ES(1+9)$	$ES(5+5)$
5×5	732.12(11)	1352.51(18)	583.30(8)	1098.77(32)
15×15	787.45(8)	1158.33(8)	521.46(2)	882.67(7)
50×1	813.79(11)	1416.00(9)	642.09(2)	1151.80(26)

Tabulka 6.8: *Průměrný a minimální počet generací k nalezení nejkratší cesty k vytyčenému cíli. Bylo použito 100 testovacích běhů pro každou konfiguraci. Všechny běhy ohraničené 11000 generacemi byly úspěšné a našli vždy nejkratší cestu délky 9. Modře jsou znázorněny nejlepší řešení napříč mutacemi a evolučními strategiemi.*

6.4.4 Nejkratší cesta s překážkou

Model bludiště byl upraven podle návrhu z kapitoly 6.4.1. Optimální cestu nalezenou v testech s hledáním přímé cesty nyní narušíme překážkou. Tento fakt bude klást na agenta zvýšené nároky.

Objevovaly se dva druhy řešení, jedno z řešení hledalo stále nejkratší cestu bez ohledu na překážku. Druhé při hledání nejkratší cesty respektovalo zakázané pole s překážkou. Na obrázku 6.16 jsou dvě obvyklé trajektorie pohybu úspěšného agenta.



Obrázek 6.16: *Dvě nejobvyklejší nejkratší trasy, kterými agent našel cíl.*

V tabulce 6.9 je znázorněno procentuální zastoupení vyevolvovaných řešení, která respektovala překážku a při hledání nejkratší cesty se jí vyhýbala. Ukázalo se, že nejlépe se s problémem vypořádaly lineární chromozomy, které mají $n_r = 1$ (čímž dochází k přirozené redukci intronů v řešení). Pokud klademe důraz na kvalitu a ne na rychlost nalezení řešení, výsledky pro vyšší mutace opět generovaly horší řešení.

$n_c \times n_r$	10% mutace		20% mutace	
	$ES(1+9)$	$ES(5+5)$	$ES(1+9)$	$ES(5+5)$
5×5	96.00%	77.00%	96.00%	77.00%
15×15	97.00%	85.00%	96.00%	83.00%
50×1	100.00%	86.00%	99.00%	86.00%

Tabulka 6.9: *Tabulka vyjadřuje procentuální zastoupení případů, kdy došlo k nalezení cíle a vyhnutí se překážce. Bylo použito 100 testovacích běhů pro každou konfiguraci. Modře jsou znázorněny nejlepší řešení napříč mutacemi a evolučními strategiemi.*

To potvrdila i tabulka D.1. Ta se vztahuje k průměrné délce trajektorie pro správné řešení vzniklé po 11000 generacích, které se vyhýbá překážce. Dodejme, že všem konfiguracím se podařilo nalézt také nejkratší možnou cestu, jejíž délka byla 13.

6.4.5 Nejkratší uzavřená cesta

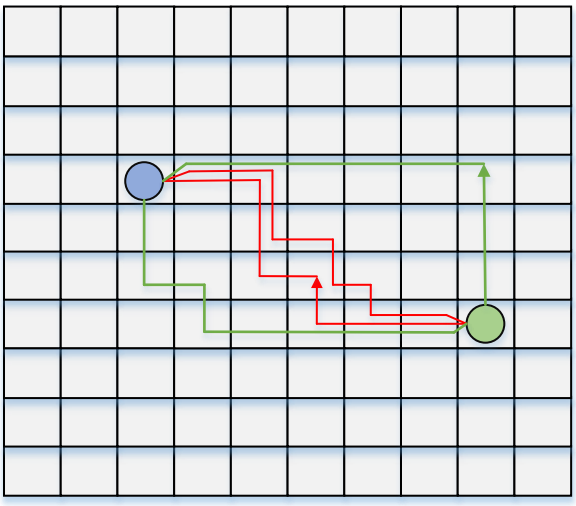
Uzavřená cesta představuje nejnáročnější úlohu ze sady úloh vztahující se k bludišti. Její složitost spočívá v podúlohách, které se agent musí naučit. Nejprve musí nalézt nejkratší cestu ke zvolené cílové pozici. V cílové pozici musí chápat, že se má vrátit zpět do počáteční pozice a přitom musí opět urazit co nejkratší cestu. Je zřejmé, že evolvovaná matematická funkce je velmi komplexní. Pro zjednodušení úlohy jsou vstupem neuronové sítě při vyhodnocování nejenom dva signály polohy agenta, ale i signál i_r.

Pro všechny statistické běhy se nám podařilo nalézt agenta, který splnil požadovanou úlohu. V tabulce 6.10 jsou znázorněny průměrné délky požadovaného typu trasy bludištěm. Můžeme konstatovat, že všechny konfigurace v rámci zvolené strategie a míry mutace byly srovnatelné a lišili se jen málo.

$n_c \times n_r$	10% mutace		20% mutace	
	$ES(1+9)$	$ES(5+5)$	$ES(1+9)$	$ES(5+5)$
5×5	22.38(18)	22.86(18)	22.64(18)	22.72(18)
15×15	21.38(18)	22.34(18)	22.84(18)	23.28(20)
50×1	21.80(18)	22.60(18)	23.09(18)	23.08(18)

Tabulka 6.10: *Průměrná a minimální délka úspěšně vykonané cesty. Formát zápisu je: průměrná délka trasy(minimální dosažená délka trasy). Minimální dosažitelná délka trasy v daném bludišti byla 18. Bylo použito 100 testovacích běhů pro každou konfiguraci. Všechny běhy ohraničené 11000 generacemi byly úspěšné. Modře jsou znázorněny nejlepší řešení napříč mutacemi a evolučními strategiemi.*

Obrázek 6.17 ukazuje nejčastější tvar trajektorií pohybu agenta v bludišti. Pro tuto nejnáročnější úlohu byla připravena i ukázka konkrétní instance neuronové sítě na obrázku D.1. Otevírají se tak nové možnosti, jak úlohu upravit. Jedním ze směrů je stimulovat chování agenta pomocí ohodnocení tak, aby hledal nejkratší trasu a zároveň k tomu použil co nejméně změn směrů. Druhý způsob úpravy zadání úlohy spočívá ve snížení počtu signálů a zapojení více neuronových sítí, které dohromady vytvoří komplexnější model agenta. Oba tyto návrhy byly zanechány ve stádiu konceptu.



Obrázek 6.17: *Dvě nejobvyklejší nejkratší trasy, kterými agent našel uzavřený okruh mezi cílem a počátkem.*

6.5 Pohyb v bludišti pomocí *Developmental Network*

6.5.1 Popis modelu a jeho použití

V předchozím modelu bludiště popsaném v kapitole 6.4.1 se nám podařilo pomocí rekurentní neuronové sítě plnit základní úlohy. Síť přitom dostávala velmi ucelenou informaci o její aktuální poloze v bludišti, stejně tak jako informaci o míře splnění cílů. Model bludiště navržený pro *Developmental Network* (4.1) má za cíl snížit počet výstupních signálů modelu na jeden, jehož definiční obor je z intervalu $< 0; 255 >$. Klademe důraz na to, aby si samotná síť (agent), nikoliv za přímé pomoci modelu, vytvářela prostorovou představu o tom, kde se nachází a co má za úkol.

Model, stejně jak tomu bylo u modelu bludiště pro přímé zakódování, definuje proměnnou $E = 200$, která je při každém kroku ohodnocovaného agenta dekrementována (o 1) a určuje tak dobu jeho setrvání v bludišti. Ohodnocení f_v je inicializované počátečním nastavením na $f_i = 0$ a při každém kroku agenta je inkrementováno (o 1). Je to proto, že v nastavení této úlohy nám jde především o splnění cílů, jako je nalezení koncového bodu a návrat zpět do počátku a neklademe již takový důraz na délku samotné trajektorie. Inkrementace f_v v podstatě kladně ohodnocuje síť, která se dokázala rozvinout do takové podoby, že je schopná komunikovat s okolím a tím agentem pohybovat v bludišti.

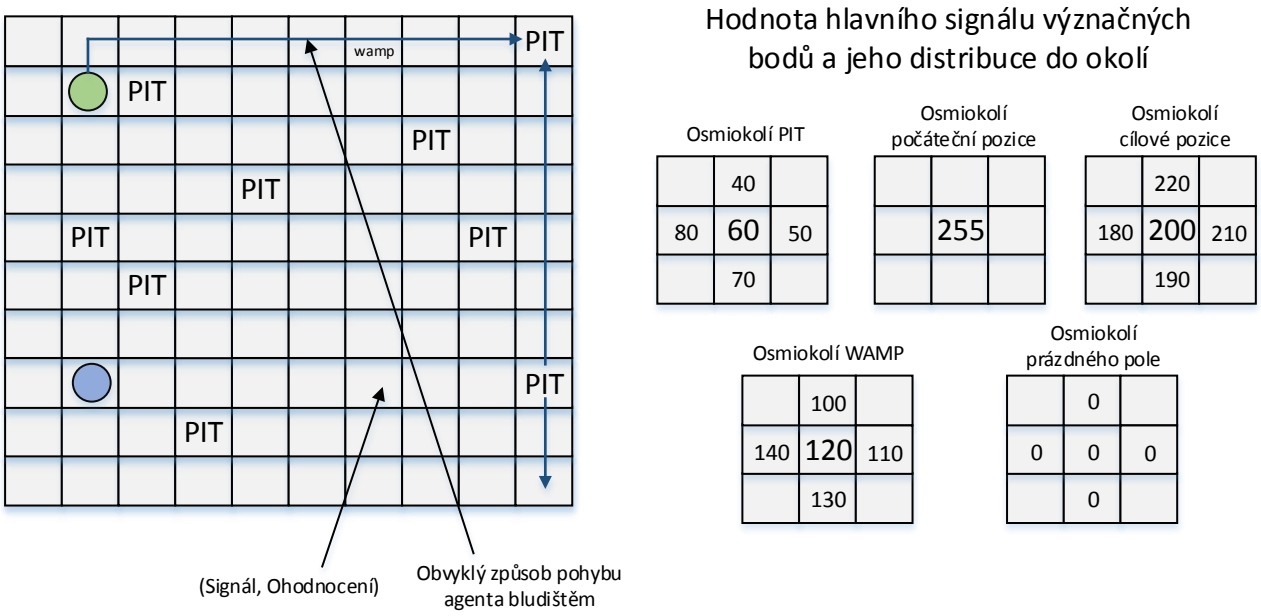
Uvědomme si, že *Developmental Network* představuje alternativu nepřímého zakódování, kdy právě během vyhodnocování modelem dochází k změnám tvaru topologie. U přímého zakódování vyhodnocování probíhalo nad tvarově neměnnou plně vygenerovanou topologií.

Jelikož chybí signály o informaci splnění daného podcíle agentem, při splnění podcíle vhodně model navýší hodnotu E tak, abychom prodloužili dobu interakce neuronové sítě s prostředím a dovolili ji adaptovat se pro splnění následného podcíle. Se splněním podcíle se pojí také zvýšení f_v, abychom odlišili genotypy, které jsou úspěšné. Na druhou stranu, můžeme E potažmo f_v významně snižovat v případě, že síť neprovádí požadovanou úlohu.

Bludiště také nebude prázdné tak, jak jsme byli zvyklí v případě 6.4.1. Vycházíme

z tzv. *Wampus world*. Jedná se o bludiště, ve kterém je řada statických nástrah. Jak je vidět na obrázku 6.18, pole bludiště můžeme rozčlenit na 5 typů (odpovídajících 5 hlavním signálům).

- `Počáteční pozice` S_p - Výchozí bod vyhodnocování, jako jediný nedistribuuje svůj vliv do okolí.
- `Cílová pozice` S_c - Pozice, ze které se agent musí vrátit zpět do S_p. Pokud se do ní agent dostane, navýší se E a f_v, aby se agent mohl vrátit zpět do počáteční pozice.
- `WAMP` S_{WAMP} - Představuje hlavní nástrahu, pokud se agent dostane na pole S_{WAMP}, jeho E je snížena o 60%.
- `PIT` S_{PIT} - Představuje dílčí nástrahu, energie je v případě přechodu přes toto pole snížena o malou kladnou konstantu. Jedná se především o prvek určený pro orientaci v bludišti.
- `Prázdné pole` S_b - Základní pole bludiště.



Obrázek 6.18: *Bludiště s rozmístěním nástrah. Obsahuje ukázku (v podobě trajektorie průchodu bludištěm) typického špatného chování získaných řešení. V pravé části je výsek bludiště s detailním popisem hodnot hlavních a distribuovaných signálů.*

Každý typ políčka (představující hlavní signál) distribuuje svůj signál i na východ, sever, jih a západ od své pozice. V případě, že je stejné políčko zasaženo různými typy signálů, vybere se pro něj pouze jeden dle následujících pravidel

- Signál z S_b je nahrazen kterýmkoliv hlavním nebo distribuovaným signálem.

- Hlavní signál, vyjma signálu z S_b, je vždy upřednostňován před distribuovaným signálem.
- Pokud na jedno políčko vychází více distribuovaných či více hlavních signálů, o jeho obsazení rozhodne preferenční relace mezi 5 popsanými typy:

$$S_b < S_{PIT} < S_c < S_{WAMP} < S_p$$

.

Pokud je agent na vybraném políčku, právě signál z políčka je pro něj jedinou informací, kde se nachází (např. 100 znamená, že jižně od aktuální pozice se nachází WAMP).

Důvodem tak velkého množství signálů v bludišti je umožnění agentovi, aby si na základě různorodých vstupů z bludiště mohl vytvořit svou vlastní paměťovou mapu a orientovat se správně bludištěm. V důsledku toho by mohla vznikat robustní řešení, která by zadanou úlohu mohla vykonat i v případě, že nezačnou z předem známé pozice tak, jak jsme navrhovali v kapitole 6.4.

V kapitole 4.1 je popsáno také vytvoření výstupní hodnoty z intervalu $< 0; 255 >$. Model tuto hodnotu příjme a opět na jejím základě vybírá index směru pohybu dle logiky popsané v modelu bludiště pro přímé zakódování *(viz. výraz 6.2)*. Index směru je dán jako

$$i = \frac{tr(Ao)}{255}$$

6.5.2 Hlavní problémy při evoluci řešení

Našim cílem bylo evolvovat nastavení CGP chromozomů tak, aby během komunikace agenta s modelem vznikala neuronová síť, která se dostane v bludišti do cílové pozice a zpět na počáteční pozici. Přitom se jí podaří vyhnout všem nástrahám. Řešení nastíněné v [6] se potýkalo s mnoha potížemi a nepodařilo se nám dosáhnout úspěchu. Proto popišme základní problémy a několik postupných úprav, kterými jsme se snažili řešení zlepšit. Dodejme, že jsme použili implicitní hodnoty nastavení všech parametrů, jak je popsáno v kapitolách 6.4.1 a 4.1.

Prvním problémem byl proměnný počet vstupů a výstupů sítě vycházející z návrhu. Ten se měnil během komunikace agenta s modelem. Každá změna (odstranění, přidání) či pohyb vstup po mřížce zásadně měnil způsob přinášení signálu k neuronům sítě. Častým jevem pak byla vstupně výstupní exploze, kdy během několika kroků vyhodnocování několikanásobně vzrostl počet vstupů a výstupů sítě. To mělo za následek zahlcení celé topologie, která začala nekontrolovaně růst. Opačným případem byl stav, kdy zanikla všechna vstupní axonální zakončení. To naopak způsobilo vlivem působení chromozomů životního cyklu postupný kolaps a zánik celé sítě.

Z hlediska výpočetní náročnosti byly problematičtí jedinci, kde proběhl abnormální nárůst počtu komponent díky vstupně výstupní explozi. Tyto jedince jsme začali odfiltrovávat, aby algoritmus končil v časově dosažitelném horizontu. Díky častým problémům se vstupy a výstupy sítě jsme se rozhodli nastavit jejich počet, stejně tak jejich pozici v mřížce, za konstantní. I z hlediska počtu kroků evoluce k dosažení hledaného řešení se tak snížily celkové nároky a zakódování sítě obecně zmenšilo svůj stavový prostor pro hledání nejlepšího řešení.

Ne zcela detailně bylo v [6] popsáno, jak správně zpracovat hodnoty potenciálu šířícího se sítí. Autor nejasně popsal, zda se procesu účastní vždy jen aktivní komponenty, nebo

i komponenty neaktivní a pojmy zaměňoval. My jsme ve výpočtech vždy uvažovali pouze komponenty aktivní. Stejně tak nebyly úplně zřejmé způsoby úpravy hodnot *state-factoru*.

V práci také nebylo přesně popsáno prahování (škálování) hodnoty h, která rozhodovala o odstraňování a přidávání komponent. Proto jsme jednoznačně toto prahování definovali. Správné nastavení prahů omezilo početní expanze komponent v mřížce.

Posledním problematickým bodem byly samotné *CGP* chromozomy. 7 chromozomů, každý o velikosti $n_c = 100$, byly mutovány s nízkým procentem mutace, vždy na konci celé komunikace agenta s modelem (samozřejmě jen v případě rodičů). Tím, že je použit pouze operátor prosté mutace, je velmi obtížné zajistit, aby se na základě chování těchto chromozomů vybudovala kompaktní neuronová síť v mřížce. Pro zmenšení stavového prostoru možných tvarů fenotypů vzniklých z chromozomů jsme se rozhodli jejich velikost snížit na $n_c = 30$.

Chromozomy upravující šíření potenciálu sítí měly konstantní počet vstupů, ale proměnný počet vstupních hodnot, jelikož vstupy tvořily atributy komponent, které se během života jedince mohli přidávat a odstraňovat. To nepřispělo ke správné konvergenci evolučního procesu.

S konvergencí k lepším řešením souvisí i způsob šíření signálu sítí. Téměř ve všech případech postupného vybírání komponent pro určitou operaci (například výběr aktivních dendrálních výběžků jednoho dendritu při distribuci potenciálu sítí) se komponenty vybírají v náhodném pořadí. Náhodně se také nové komponenty vkládají do mřížky. Stejně tak se s komponentami na základě změn parametru r náhodně pohybuje. Tato náhodnost vede k jedné nepříznivé vlastnosti. Pokud ohodnotíme jedince s konkrétní sadou chromozomů a konfigurací rozložení komponent v mřížce ohodnocením f_v neznamená to, že jeho opětovné ohodnocení dosáhne při naprosto stejné výchozí konfiguraci opět stejné hodnoty f_v. Proto se často při strategii $ES(1 + 9)$ stávalo, že jsme již měli takové nastavení *CGP* chromozomů a počátečního rozmístění komponent v mřížce, že se síť, resp. agent, dokázal úspěšně pohybovat v bludišti. V následující generaci toto nastavení dostalo velmi nízké ohodnocení například kvůli náhodnému přidání jednoho neuronu do mřížky, čímž se celá topologie destabilizovala. Potom byla potlačena hlavní myšlenka evolučního algoritmu o distribuci nejlepšího řešení napříč generacemi, jak je popsáno v kapitole 2.2.

Navržené řešení problému spočívalo v uchování si nejvyšší dosažené fitness hodnoty f_v konkrétního jedince přes všechny generace, ve kterých se vyskytl. Potom hodnota fitness vypovídala více o potenciálu konfigurace vygenerovat kvalitní řešení, než o samotné kvalitě vygenerovaného jedince. Druhá (nerealizovaná) možnost řešení je dávkové, nikoliv sekvenční provádění algoritmu. Znamenalo by to další paměťové nároky z hlediska uchování hodnot mezivýsledků, které by se aplikovaly až po zpracování průchodu jednoho signálu z modelu celou sítí. Takové provádění algoritmu by si vyžádalo více zásadních úprav návrhu a ponecháme ho jako jedno z východisek.

Autoři v [6] použili konkrétní neměnné počáteční nastavení rozložení komponent v mřížce. Správné počáteční rozložení může výrazně ovlivnit chování celého algoritmu. Bohužel jejich nastavení ve větším detailu nepopsali. V práci pouze představili nepříliš přehledný obrázek vzhledu iniciálního nastavení jejich mřížky. Tuto konfiguraci jsme se pokusili interpetovat a experimentovat s ní. Nastavení mřížky snažící se napodobit to z práce v [6] lze nalézt na *CD* v *csv* souboru s názvem `originalTopoFile.model`.

Pro nejlepší počáteční nastavení jsme proto alternativně vygenerovali náhodně 10000 konfigurací počátečního nastavení a populaci v první generaci sestavili z nejlépe ohodnocených jedinců. Z těchto nastavení vznikly dvě uspořádání mřížky a komponent, se kterými jsme experimentovali. Počáteční nastavení jsou uložena na přiloženém *CD (viz. příloha E)*

pod názvy `bestTopoFile1.model` a `bestTopoFile2.model`.

Posledním důležitým bodem je správné umístění počáteční a cílové pozice v modelu bludiště. O správném rozložení hovoříme proto, že ne všechny vzájemné pozice těchto bodů v bludišti mají stejnou vypovídací hodnotu. Představme si příklad z obrázku 6.18 s tím rozdílem, že úkolem agenta je pouze dostat se do cílové pozice v pravém horním rohu bludiště. U špatně vyevolvovaných sítí se běžně stávalo, že jejich výstup byla hodnota blízká 0. Z logiky výběru směrů pohybu by se agent pohyboval nejprve na sever, po dosažení hrany bludiště na východ, až by se dostal do cíle. Nefunkční řešení bychom potom museli prohlásit za úspěšné. Proto jsme se takovýmto testovacím nastavením bludiště vyhnuli.

I po všech popsaných optimalizacích jsme nepozorovali žádnou konvergenci k průměrně lepšímu ohodnocení jedinců v populaci, jakožto ani konvergenci k lepším ohodnocením nejlepších jedinců v populaci. V obrázku 6.18 je znázorněno použité nastavení bludiště s nejobvyklejší trasou agenta, který nejčastěji zůstal v oblasti pravého horního rohu bludiště. Chování nezměnilo ani záměrné umístění signálu $WAMP$ přímo do trajektorie agentova pohybu. Pole s hodnotou $WAMP$ nejradikálněji snižovalo ohodnocení takto se pohybujícího jedince.

V této sekci dále neuvádíme další experimenty. V závěru práce uvádíme některé možné úlohy vycházející z nezdaru při použití *Developmental Network*.

Kapitola 7

Závěr

V práci jsem se zabýval koevolucí kartézského genetického programování a neuronových sítí. Teoretická část práce byla proto věnována popisu obou technik. Navrhl jsem možné způsoby koevoluce genetického programování a neuronových sítí. Jednalo se o přímé zakódování rekurentní alternativy Jordanovy sítě a obecné dopředné sítě do chromozomů genetického programování. Druhá část návrhu popisovala nepřímé zakódování *Developmental Network*. Pomocí těchto návrhů byla implementována knihovna `cgpdnn`, u které byl kladen důraz na přenositelnost a modularitu. S knihovnou jsem provedl další výzkum a experimenty.

Pro experimenty jsem vybral tři úlohy. Úloha detekce buněk rakoviny představuje typickou úlohu zaměřenou na klasifikační schopnosti výsledné sítě. Model vyvažování tyče na vozíku je jedna ze základních benchmarkových úloh pro testování sítě jako kontroléru. Poslední úlohou byl model bludiště, ve kterém síť v podobě agenta plnila zadané úkoly. Koevoluční řešení problému bludiště pro přímé zakódování neuronové sítě přitom představovalo nově prezentovaný problém, pro který neexistuje srovnatelná studie. Postupně popíši výsledky plynoucí z experimentů.

Nejprve se zaměříme na výsledky s přímým zakódováním sítě. Nepřímé zakódování sítě se bude týkat pouze experimentů s bludištěm.

Jedním z hlavních cílů pro přímé zakódování sítě bylo identifikovat nejlepší nastavení parametrů výpočtu co do velikosti chromozomů kódujících síť, vlivu mutací a použitých evolučních strategií. Jedním z testovaných konceptů byla pak mnou navržená metoda využití subpopulací. U všech experimentovaných modelů jsem z hlediska konfigurace dospěl ke stejným závěrům. Lepších výsledků dosahovaly nastavení s nižšími hodnotami n_c obsahují v poměru ke své velikosti menší počet intronů. Tomu přispívala i linearizace chromozomu pomocí $n_r = 1$. Vyšší míra mutací, stejně tak jako použití subpopulací, řešení nezlepšovalo. Vyšší míra mutace naopak způsobovala konvergenci do horších lokálních optim úloh. Jedním z možných pokračování výzkumu v této částí mé práce by bylo sledování vlivu velikosti intronů na chování fenotypu, resp. zjišťování vztahů mezi velikostí chromozomu z hlediska počtu jeho uzlů, jejich reálným zastoupením ve fenotypu a chováním fenotypu v reálném prostředí.

V úloze detekce buněk rakoviny jsem se na rozdíl od referenčních prací zaměřil na popis způsobu učení v rámci evoluce zakódování dopředné sítě a detekci časného zastavení. Experimentálně jsem prokázal, že modely schopné generalizace nevznikají až v posledních generacích řešení. Naopak dochází k přeučování. Popsal jsem také negativní vliv mutace na vývoj přesnosti řešení na testovací množině během evoluce.

Výsledné přesnosti evolvovaných sítí přesahovaly v průměru 98% na testovací množině, kdy během pouhých 20 generací měla evolvovaná síť přesnost na trénovací i testovací

množině vyšší než 80%. Konkrétní nejlepší vybrané instance řešení v časně zastaveném procesu evoluce přitom vždy klasifikovaly se 100% přesností maligní formy rakoviny. Koevoluční technika byla v úloze detekce rakoviny srovnatelně úspěšná s porovnávanými metodami.

V úloze vyvažování tyče vozíkem jsem se zaměřil na popis robustnosti vzniklých řešení a to jak u rekurentní, tak dopředné alternativy evolvované sítě. Robustnost představuje schopnost sítě vyvážit i tyč vychýlenou z počáteční pozice. Učení přitom probíhalo s modelem, jehož iniciální pozice tyče byla vyvážená. Zjistil jsem, že nejlepších výsledků dosahují nad modelem s iniciální rovnovážnou pozicí tyče řešení, která byla evolvovaná ve více generacích, než dosáhla požadovaného řešení. Jsou pak schopna lépe reagovat na extrémní pozice tyče vzhledem ke svislé ose.

Druhou sadou experimentů jsem se snažil ověřit vliv použité fitness funkce na vlastnosti vzniklého řešení. Vhodně navrženou metrikou pro fitness funkci jsem dokázal evolvovat řešení, která eliminovala vychylování tyče během vyvažování. Podařilo se mi také implicitně vytvářet robustní řešení. Tyto experimenty prokázaly, že jen za pomoci správného ohodnocení řešení lze evolvovat diametrálně odlišná řešení, která při použití konvenčních neuronových sítí vyžadují expertní znalosti návrháře takového modelu.

Poslední zkoumaný model pro přímé zakódování rekurentní sítě byl mnou navržený model bludiště. V něm jsem dále rozvinul myšlenku, že samotné ohodnocení dokáže s naprosto stejného modelu a konfigurace chromozomů evolvovat sítě s odlišným chováním. Podařilo se mi evolvovat agenta, který hledal nejkratší cestu k cíli bludiště. Mým záměrem bylo pokusit se definovat úlohu, která má více cílů. Správným nastavením fitness funkce jsem pak generoval řešení, která se dokázala vyhnout překážce v bludišti a zároveň nalézt nejkratší cestu k cíli nebo absolvovat uzavřenou cestu z počátku do cílové pozice a zpět.

Dalším postupným krokem bylo minimalizovat množství informace, kterou model agentu poskytuje. Vycházel jsem z velmi komplexního konceptu *Developmental Network*. Tato síť byla autory navržena tak, aby si vytvářela paměťový model prostředí, ve kterém se pohybuje. Tak se síť řídící agenta například dostala do cílové pozice i v případě, že hledání nezačínalo z pevně dané počáteční pozice, pomocí níž probíhalo učení. Bohužel se nám nepodařilo takové chování získat. Přesto experimenty s tímto komplexním modelem vedou k dalším možným směrům výzkumu.

Hlavním možným pokračováním v této práci vidím v navržení méně komplikovaného konceptu reflektující získané informace z práce s *Developmental Network*. Jednu instanci řešení by představovala skupinu společně evolvovaných neuronových sítí, které by svou kooperací emulovaly komplexně se chovající systém.

Výsledky práce z oblasti detekce buněk rakoviny byly také prezentovány na studentské konfenreci *STUDENT EEICT 2014*, kde byly oceněny 1.místem v kategorii *Inteligentní systémy* a cenou hlavního sponzora konference, společností *Honeywell*.

Literatura

[1] Ahmad, A. M.; Khan, G. M.; Mahmud, S. A.; aj.: Breast cancer detection using cartesian genetic programming evolved artificial neural networks. *GECCO '12: Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation*, 2012: s. 1031–1038, 910122.

[2] Brownlee, J.: The Pole Balancing Problem - A Benchmark Control Theory Problem. Technická zpráva, 2005.
URL http://researchbank.swinburne.edu.au/vital/access/services/Download/swin:7595/SOURCE1

[3] Eiben, A. E.; Smith, J. E.: *Introduction to Evolutionary Computing*. SpringerVerlag, 2003, ISBN 3540401849, 300 s.

[4] Holland, J. H.: *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992, ISBN 0262082136, 228 s.

[5] Jordan, M. I.: Artificial Neural Networks. kapitola Attractor Dynamics and Parallelism in a Connectionist Sequential Machine, Piscataway, NJ, USA: IEEE Press, 1990, ISBN 0-8186-2015-3, s. 112–127.

[6] Khan, G. M.; Miller, J. F.; Halliday, D. M.: Evolution of Cartesian Genetic Programs for Development of Learning Neural Architecture. *Evolutionary Computation*, ročník 19, č. 3, 2011: s. 469–523, ISSN 1063-6560.

[7] Khan, M. M.; Ahmad, A. M.; Khan, G. M.; aj.: Fast learning neural networks using Cartesian genetic programming. *Neurocomputing*, ročník 121, 2013: s. 274–289, ISSN 0925-2312.

[8] Kvasnička, V.; Pospíchal, J.; Tiňo, P.: *Evoluční algoritmy*. STU, 2000, ISBN 80-227-1377-5, 215 s.

[9] Miller, J. F.; Harding, S.: GECCO 2012 tutorial: cartesian genetic programming. In *GECCO (Companion)*, ACM, 2012, ISBN 978-1-4503-1178-6, s. 1093–1116.

[10] Prata, S.: *Mistrovství v C++*. Brno: Computer Press, třetí vydání, 2007, 1119 s.

[11] Ranzato, M.; Krizhevsky, A.; Hinton, G. E.: Factored 3-Way Restricted Boltzmann Machines For Modeling Natural Images. In *AISTATS*, *JMLR Proceedings*, ročník 9, editace Y. W. Teh; D. M. Titterington, JMLR.org, 2010, ISSN 1532-4435, s. 621–628.

[12] Sammut, C.; Webb, G. I. (editoři): *Encyclopedia of Machine Learning*. Springer, 2010, ISBN 978-0-387-30768-8, 1059 s.

[13] Sekanina, L.; Vašíček, Z.; Růžička, R.; aj.: *Evoluční hardware: Od automatického generování patentovatelných invencí k sebemodifikujícím se strojům*. Edice Gerstner, Academia, 2009, ISBN 978-80-200-1729-1, 328 s.

[14] Švarc, S.; Krupková, V.; Studená, V.; aj.: *Matematická analýza*. Učební texty vys. šk, Vysoké učení technické v Brně. Fakulta elektrotechniky a informatiky, 1995, ISBN 9788021405868, 117 s.

[15] Vondrák, I.: *Umělá inteligence a neuronové sítě*. VŠB-TU, 2009, ISBN 80-7078-259-5, 140 s.

[16] Yao, X.: Evolving Artificial Neural Networks. *Proceedings of the IEEE*, ročník 87, č. 9, 1999: s. 1423–1447, ISSN 0018-9219.

Příloha A

Tabulka symbolů

Tabulka symbolů použitých v kapitolách 4 a 6.

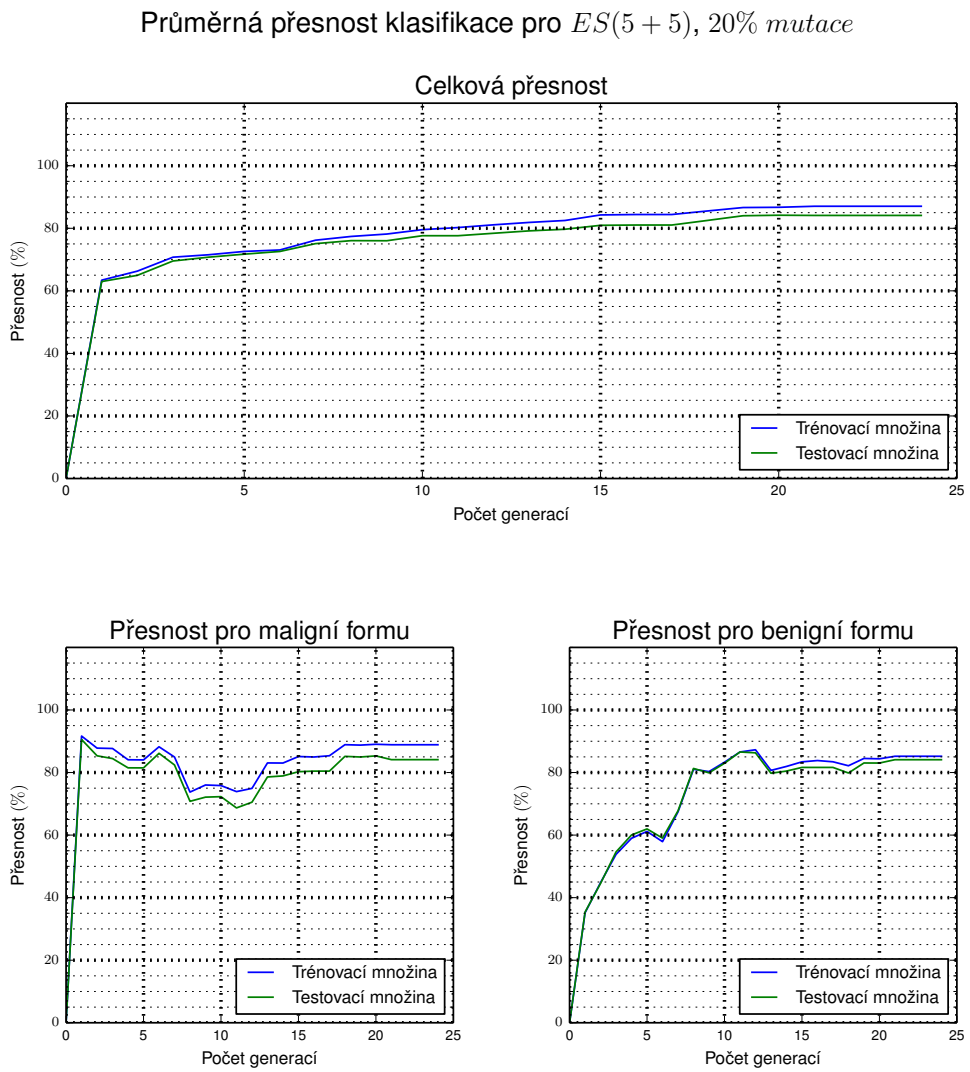
Symbol	Význam
Symboly vztahující se k návrhu *Developmental Network*	
az	Index vztahující proměnnou k axonálnímu zakončení
a	Index vztahující proměnnou k axonu
dv	Index vztahující proměnnou k dendrálnímu výběžku
d	Index vztahující proměnnou k dendritu
s	Index vztahující proměnnou k somě
G_l	Počet řádků mřížky
G_c	Počet sloupců mřížky
N_{az}	Maximální počet axonálních zakončení při inicializaci axonu
N_d	Maximální počet dendritů při inicializaci neuronu
N_{dv}	Maximální počet dendrálních výběžků dendritu při inicializaci
N_n	Maximální iniciální počet neuronů jedince v mřížce
max	Hodnota 255
$base$	Hodnota 256
h	Atribut health
sf	Atribut state-factor
r	Atribut resistance
p	Atribut potenciál
p'	Atribut potenciál ovlivněný pomocí dalších atributů
w	Atribut váhy
H_{min}	Práh pro h určující zánik komponenty
H_{max}	Práh pro h určující replikaci komponenty
R	Minimální velikost změny r, která vyvolá pohyb komponenty
P	Minimální změna potenciálu komponenty (vyjma somy), která vyvolá úpravu sf
P_{trash}	Průběžný práh pro výboj z daného neuronu
D	Konstanta přirozeného úbytku velikosti hodnoty atributů komponent

Symboly vztahující se k zakódování do CGP chromozomů	
n_r	Počet řádků chromozomu
n_c	Počet sloupců chromozomu
n_n	Arita funkcí
n_f	Počet možných indexovaných funkcí
L	L back parametr chromozomu
N_{bit}	Počet mutovaných genů chromozomů jedince
Γ	Seznam možných funkcích v uzlech chromozomu
I_{ik}	Ukazatel na výstupu daného uzlu, který je přiveden na i.vstup k. uzlu
I^*_{ik}	Hodnota výstupu uzlu referencovaného s I_{ik}
s_{ik}	Aktivita spojení mezi k. uzlem a uzlem referencovaným indexem I_{ik}
w_{ik}	Váha spojení mezi k. uzlem a uzlem referencovaným indexem I_{ik}
f_{cr}	Ukazatel do tabulky funkcí na funkci umístěnou v c.sloupci a r.řádku chromozomu
f^*_{cr}	Funkce umístěná v c.sloupci a r.řádku chromozomu
Symboly vztahující se k experimentům	
f_v	Ohodnocení jedince
I_x	Povolený interval pozice vozíku při vyrovnávání tyče
I_θ	Povolený interval vychylování tyče
N	Maximální počet kroků vyvažování vozíku
M	Obecně model prostředí, vztahuje se k právě diskutovanému modelu
$M(\theta)$	Model vyvažování s počáteční výchylkou θ
$eval$	Funkce ohodnocení jedince nad modelem
S_α	Suma korků vyvažování jedince α v testech robustnosti
E	Zbývající možný počet kroků agenta v bludišti
C, C_a, C_b	Kladné konstanty ohodnocení chování agenta v bludišti
A	Počet alternativ směrů, kterými lze v daném bodě bludiště jít
i_n, i_r	Signály polohy a splnění cílů agenta v bludišti
tr	Funkce ořezání desetinné části čísla
$S_{\text{specifikace}}$	Typ signály (nutná specifikace) v bludišti *Developmental Network*

Příloha B

Příloha experimentů s buňkami rakoviny

B.1 Graf konvergence



Obrázek B.1: *Rychlost konvergence při vysokých mutacích. Uvažujeme maximální trénovací přesnost a příslušnou maximální testovací přesnost.*

B.2 Tabulka průměrné přesnosti v bodě α

n_c	n_n	10% mutace		20% mutace	
		$ES(1+9)$	$ES(5+5)$	$ES(1+9)$	$ES(5+5)$
50	10	98.90% ; 95.30% (6505)	98.20% ; 95.90% (2520)	97.70% ; 94.60% (2161)	97.90% ; 95.20% (7110)
	25	98.30% ; 95.10% (2441)	98.20% ; 95.60% (6947)	98.00% ; 94.50% (3420)	98.40% ; 94.40% (4196)
	40	98.40% ; 94.80% (4623)	98.40% ; 96.20% (5316)	97.90% ; 95.60% (1441)	97.60% ; 95.00% (5392)
100	10	97.70% ; 94.70% (799)	98.40% ; 95.40% (4396)	98.00% ; 95.40% (7699)	98.30% ; 94.90% (6534)
	25	98.50% ; 95.00% (1112)	98.50% ; 95.40% (10158)	98.40% ; 94.60% (4849)	97.60% ; 95.10% (4498)
	40	97.00% ; 94.70% (1309)	97.80% ; 95.00% (914)	97.70% ; 95.40% (2203)	98.00% ; 95.50% (8793)
200	10	98.60% ; 94.60% (8073)	98.90% ; 95.50% (2743)	98.40% ; 94.60% (8927)	97.60% ; 95.40% (2964)
	25	98.40% ; 95.80% (4247)	95.70% ; 95.90% (676)	98.40% ; 95.30% (1644)	98.00% ; 95.30% (6955)
	40	98.30% ; 93.80% (1397)	98.40% ; 95.00% (7775)	98.10% ; 94.60% (4303)	97.40% ; 95.30% (3246)
300	10	98.50% ; 95.60% (1959)	98.50% ; 95.40% (8876)	98.40% ; 94.80% (802)	97.50% ; 95.40% (1132)
	25	97.60% ; 93.70% (961)	96.50% ; 96.00% (1009)	98.40% ; 93.50% (8727)	97.90% ; 94.90% (5664)
	40	98.30% ; 94.10% (1995)	97.60% ; 95.10% (2410)	97.20% ; 94.10% (440)	97.50% ; 95.10% (8999)

Tabulka B.1: *Průměr přesnosti klasifikace na trénovací a testovací množině s průměrným počtem generací k jejímu dosažení v bodě α. Formát buňky tabulky je: přesnost na trénovací množině ; přesnost na testovací množině (počet generací). Modře jsou znázorněny nejlepší výsledky pro konfiguraci definovanou mírou mutace, evoluční strategií a n_c.*

B.3 Tabulka průměrné přesnosti v bodě β

n_c	n_n	10% mutace		20% mutace	
		$ES(1+9)$	$ES(5+5)$	$ES(1+9)$	$ES(5+5)$
50	10	98.90% ; 95.30% (6505)	98.60% ; 95.50% (7936)	98.10% ; 92.60% (3376)	98.50% ; 94.60% (8000)
	25	98.90% ; 94.10% (6543)	99.00% ; 95.00% (2478)	98.20% ; 94.00% (4839)	98.50% ; 94.30% (3405)
	40	98.50% ; 94.50% (4632)	98.90% ; 95.50% (7632)	98.50% ; 93.70% (3438)	98.50% ; 93.90% (6739)
100	10	98.90% ; 94.30% (9029)	98.80% ; 95.00% (8146)	98.10% ; 94.90% (7912)	98.40% ; 94.80% (8106)
	25	98.50% ; 95.00% (1112)	98.90% ; 95.00% (8133)	98.40% ; 94.60% (4849)	98.50% ; 93.10% (9547)
	40	98.50% ; 94.00% (6029)	98.50% ; 94.50% (8726)	98.50% ; 93.40% (3853)	98.40% ; 93.10% (2397)
200	10	98.90% ; 94.50% (8145)	99.00% ; 95.30% (2861)	98.40% ; 94.60% (8927)	98.50% ; 93.50% (8144)
	25	98.80% ; 93.80% (9787)	98.90% ; 94.70% (5249)	98.50% ; 95.20% (2895)	98.90% ; 93.70% (2199)
	40	98.60% ; 93.40% (3193)	98.90% ; 93.70% (1832)	98.40% ; 94.40% (5480)	98.40% ; 92.90% (4359)
300	10	98.50% ; 95.60% (1959)	98.90% ; 94.20% (8148)	98.50% ; 94.40% (1605)	98.10% ; 94.20% (1833)
	25	98.10% ; 92.60% (2270)	98.60% ; 94.90% (8006)	98.40% ; 93.50% (8727)	98.40% ; 91.30% (5933)
	40	98.60% ; 93.90% (2492)	98.40% ; 94.30% (2475)	98.50% ; 93.40% (3701)	98.00% ; 93.80% (8988)

Tabulka B.2: *Průměr přesnosti klasifikace na trénovací a testovací množině s průměrným počtem generací k jejímu dosažení v bodě β. Formát buňky tabulky je: přesnost na trénovací množině ; přesnost na testovací množině (počet generací). Modře jsou znázorněny nejlepší výsledky pro konfiguraci definovanou mírou mutace, evoluční strategií a n_c.*

B.4 Tabulka maximální přesnosti v bodě β

n_c	n_n	10% mutace		20% mutace	
		$ES(1+9)$	$ES(5+5)$	$ES(1+9)$	$ES(5+5)$
50	10	99.00% ; 95.50% (5650)	99.00% ; 95.50% (6744)	98.50% ; 95.00% (3553)	98.50% ; 95.00% (6915)
	25	99.00% ; 94.00% (5680)	99.00% ; 95.00% (7006)	99.00% ; 92.00% (8465)	98.50% ; 94.50% (2851)
	40	98.50% ; 95.00% (5683)	99.00% ; 95.50% (8730)	98.50% ; 94.50% (4035)	98.50% ; 94.00% (5783)
100	10	99.00% ; 94.00% (10415)	99.00% ; 95.00% (9747)	98.50% ; 92.50% (1236)	98.50% ; 95.00% (7806)
	25	98.50% ; 95.00% (1288)	99.00% ; 95.00% (7952)	98.50% ; 94.50% (5351)	98.50% ; 93.50% (6807)
	40	98.50% ; 94.50% (5804)	98.50% ; 94.50% (5042)	98.50% ; 95.00% (6107)	98.50% ; 92.50% (1584)
200	10	99.00% ; 94.50% (9862)	99.00% ; 95.50% (2999)	98.50% ; 95.00% (9137)	98.50% ; 93.50% (7070)
	25	99.00% ; 93.50% (10772)	99.00% ; 94.50% (5954)	98.50% ; 95.50% (1221)	99.00% ; 93.50% (2457)
	40	99.00% ; 93.00% (9404)	99.00% ; 94.00% (155)	98.50% ; 94.50% (5342)	98.50% ; 93.50% (4694)
300	10	98.50% ; 96.00% (2292)	99.00% ; 94.00% (9845)	98.50% ; 95.00% (949)	98.50% ; 93.00% (4358)
	25	98.50% ; 93.00% (5466)	99.00% ; 94.50% (9102)	98.50% ; 93.50% (10624)	98.50% ; 91.00% (7234)
	40	99.00% ; 93.50% (4502)	98.50% ; 94.00% (1486)	98.50% ; 93.50% (3782)	98.00% ; 95.00% (4280)

Tabulka B.3: *Maximální hodnoty přesností klasifikace na trénovací a testovací množině s příslušným počtem generací k jejímu dosažení v bodě β. Formát buňky tabulky je: přesnost na trénovací množině ; přesnost na testovací množině (počet generací). Modře jsou znázorněny nejlepší výsledky pro konfiguraci definovanou mírou mutace, evoluční strategií a n_c.*

B.5 Tabulka maximální přesnosti v bodě δ

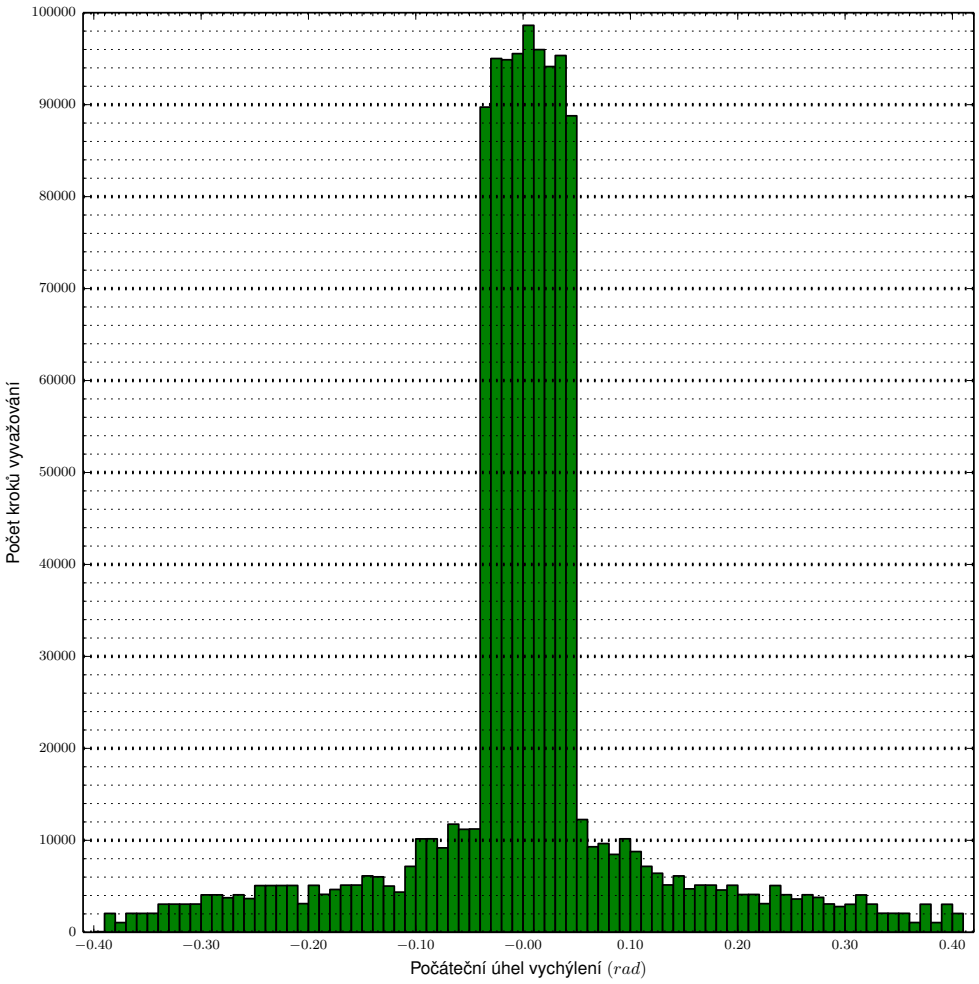
n_c	n_n	10% mutace		20% mutace	
		$ES(1+9)$	$ES(5+5)$	$ES(1+9)$	$ES(5+5)$
50	10	99.00% ; 94.50%	99.00% ; 95.50%	98.50% ; 94.00%	98.50% ; 95.00%
	25	99.00% ; 93.50%	99.00% ; 95.00%	99.00% ; 92.00%	98.50% ; 94.50%
	40	98.50% ; 94.00%	99.00% ; 95.50%	98.50% ; 92.00%	98.50% ; 94.00%
100	10	99.00% ; 94.00%	99.00% ; 95.00%	98.50% ; 92.50%	98.50% ; 95.00%
	25	98.50% ; 95.00%	99.00% ; 95.00%	98.50% ; 94.00%	98.50% ; 93.50%
	40	98.50% ; 93.00%	98.50% ; 94.50%	98.50% ; 93.00%	98.50% ; 92.50%
200	10	99.00% ; 94.00%	99.00% ; 95.50%	98.50% ; 93.50%	98.50% ; 93.50%
	25	99.00% ; 93.50%	99.00% ; 94.50%	98.50% ; 94.00%	99.00% ; 93.50%
	40	99.00% ; 93.00%	99.00% ; 94.00%	98.50% ; 93.00%	98.50% ; 93.50%
300	10	98.50% ; 93.00%	99.00% ; 94.00%	98.50% ; 95.00%	98.50% ; 93.00%
	25	98.50% ; 92.50%	99.00% ; 94.50%	98.50% ; 93.50%	98.50% ; 91.00%
	40	99.00% ; 91.50%	98.50% ; 94.00%	98.50% ; 93.50%	98.00% ; 95.00%

Tabulka B.4: *Maximální hodnoty přesností klasifikace na trénovací a testovací množině s příslušným počtem generací k jejímu dosažení v bodě δ. Formát buňky tabulky je: přesnost na trénovací množině ; přesnost na testovací množině. Modře jsou znázorněny nejlepší výsledky pro konfiguraci definovanou mírou mutace, evoluční strategií a n_c.*

Příloha C

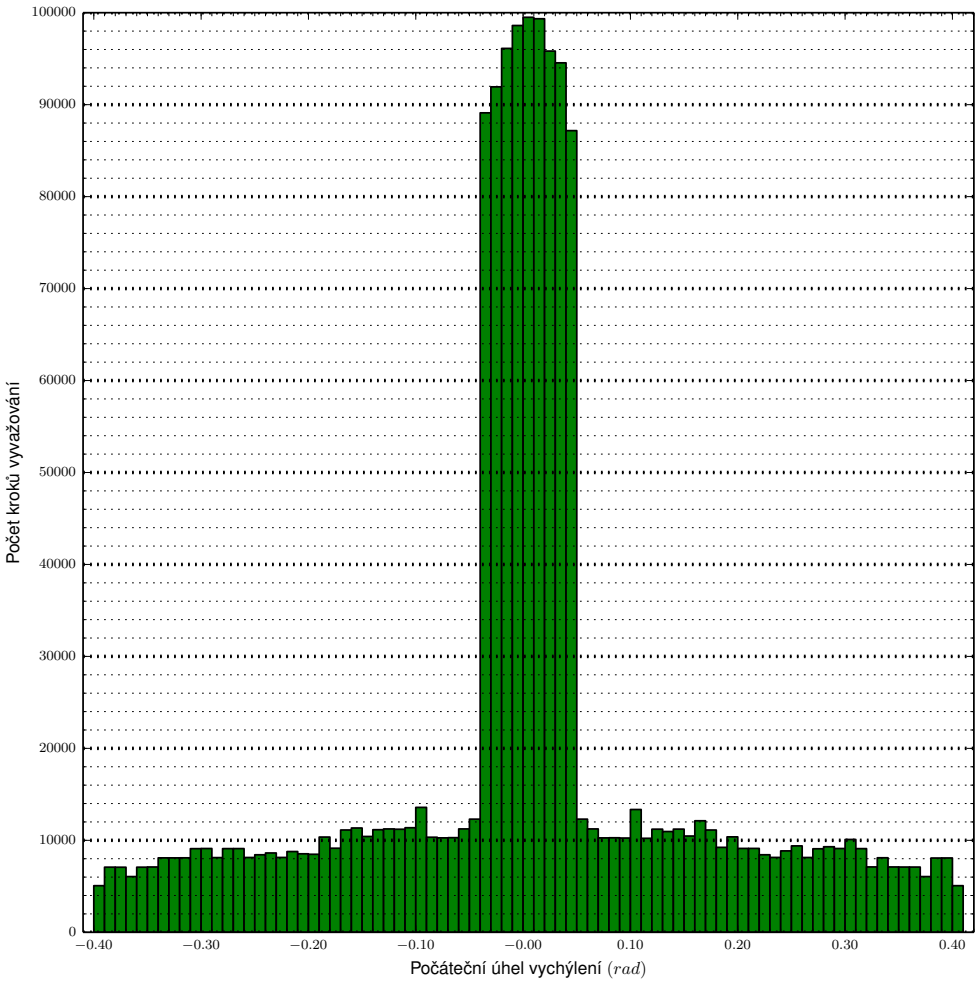
Příloha experimentů s vyvažováním tyče

C.1 Histogram počtu kroků vyvažování dopředné sítě s $n_c = 5$ a 10% mutací



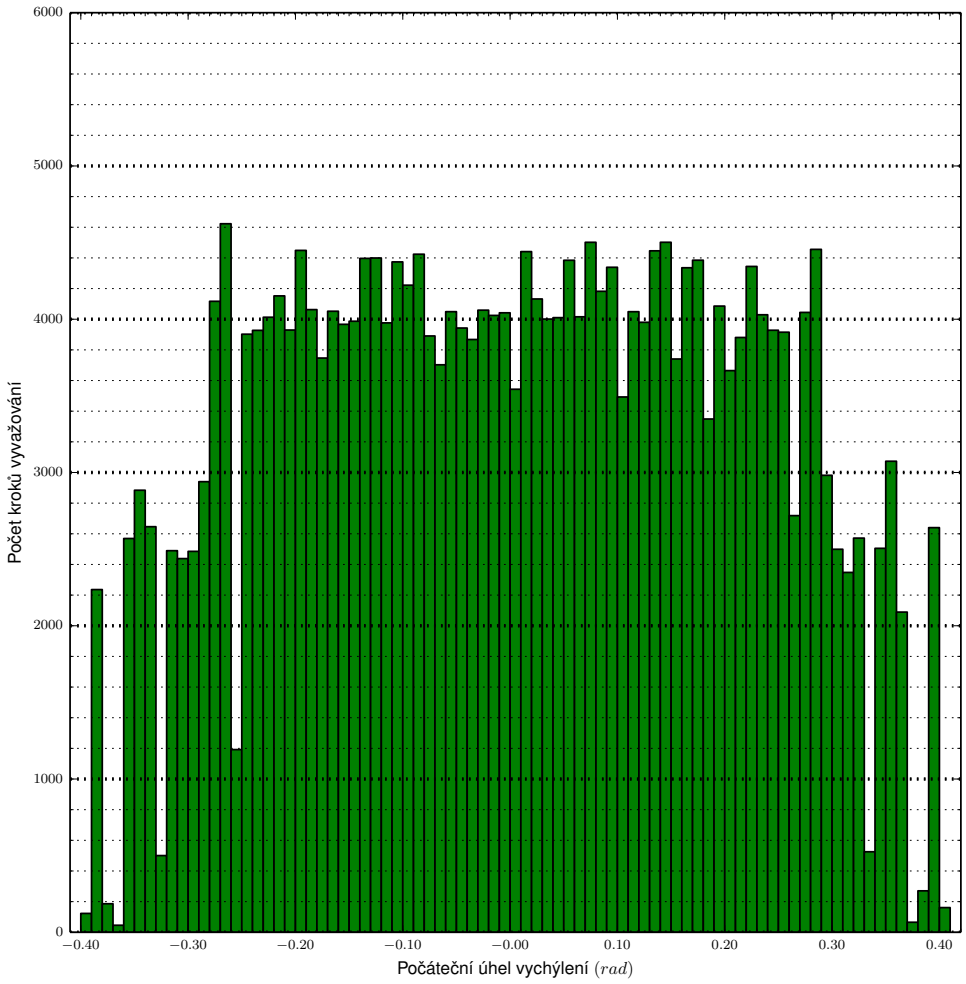
Obrázek C.1: *Pro 100 testovacích běhů vygenerujeme vždy dopřednou neuronovou síť vyvažující tyč na vozíku. Učení začíná při svislé poloze tyče, snažíme s dosáhnout 100000 kroků vyvažování. Graf zobrazuje průměrný počet kroků vyvažování tyče, jestliže natrénovaná síť započne vyvažování s již vychýlenou tyčí.*

C.2 Histogram počtu kroků vyvažování dopředné sítě s $n_c = 15$ a 10% mutací



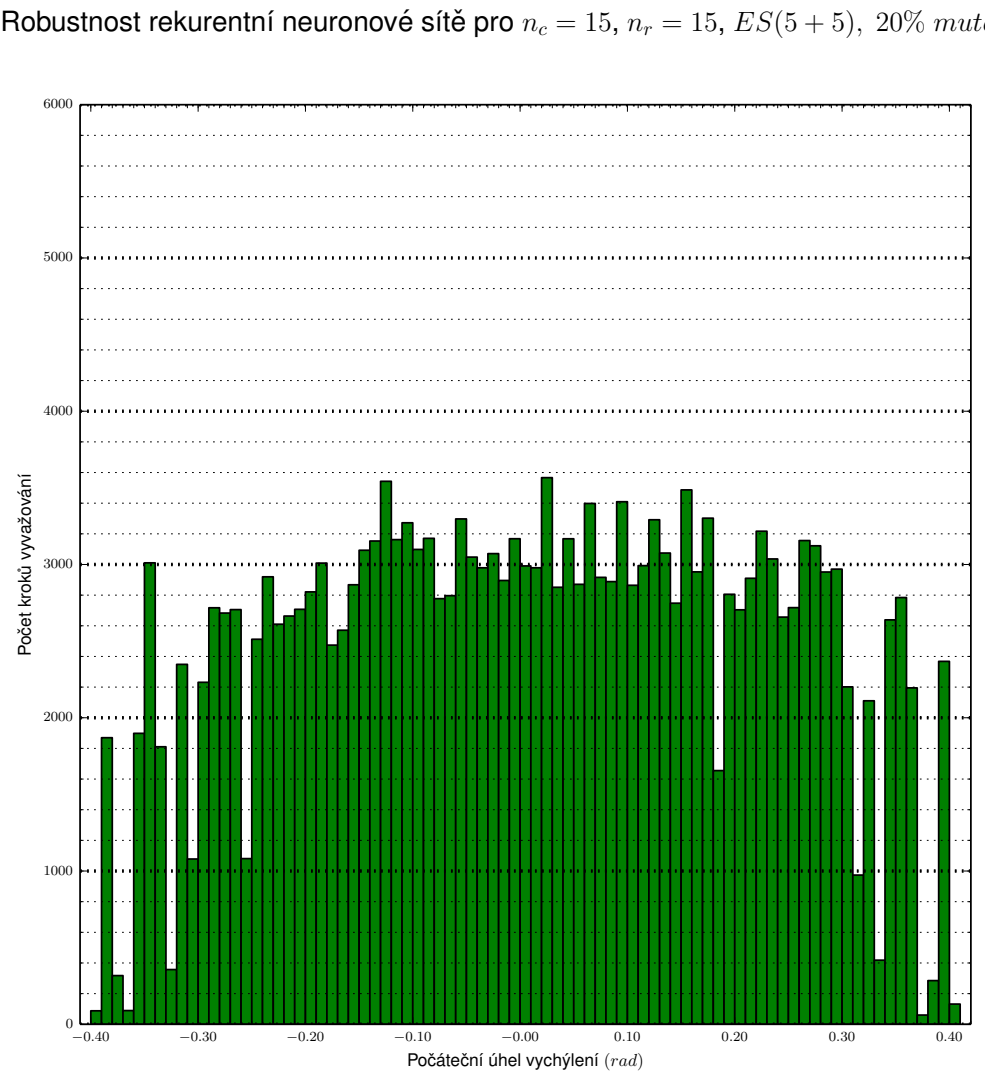
Obrázek C.2: *Pro 100 testovacích běhů vygenerujeme vždy dopřednou neuronovou síť vyvažující tyč na vozíku. Učení začíná při svislé poloze tyče, snažíme s dosáhnout 100000 kroků vyvažování. Graf zobrazuje průměrný počet kroků vyvažování tyče, jestliže natrénovaná síť započne vyvažování s již vychýlenou tyčí.*

C.3 Histogram počtu kroků vyvažování rekurentní sítě s $n_c = 15$ a 10% mutací



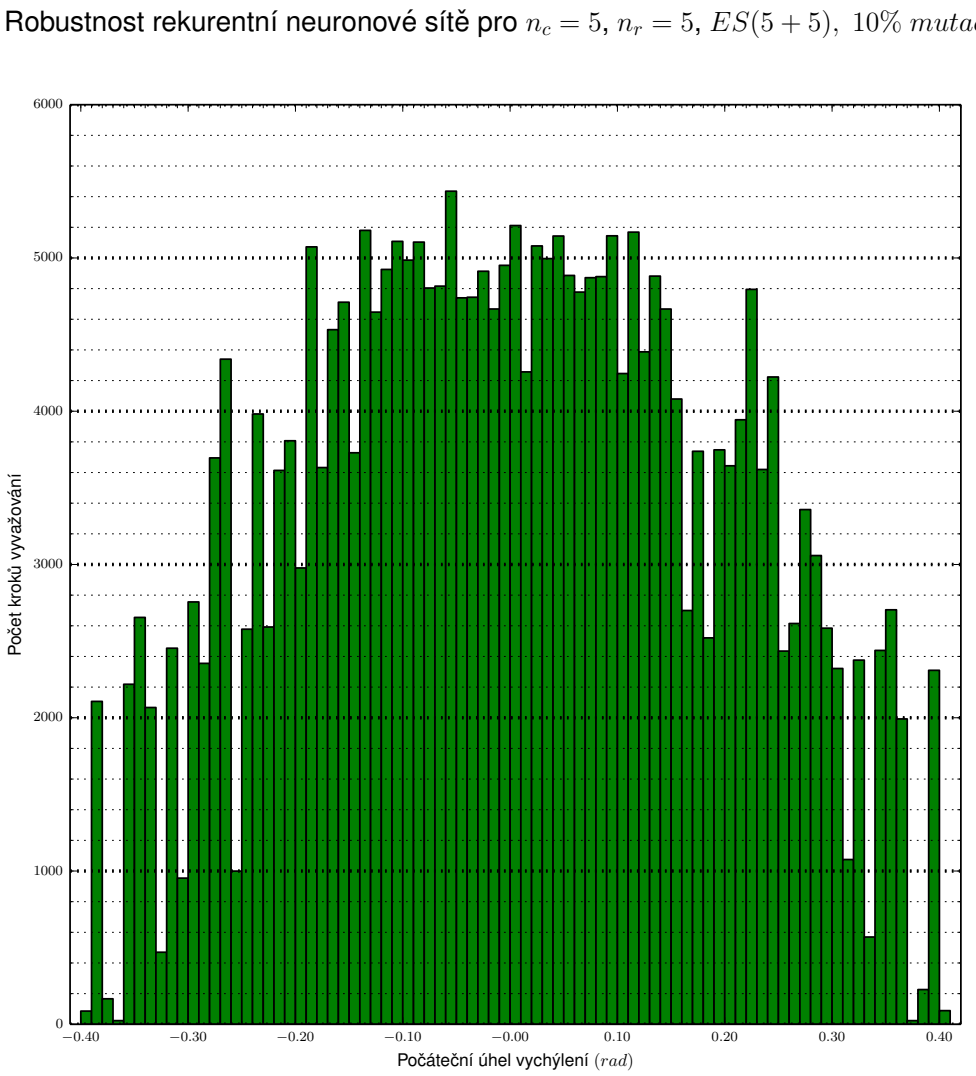
Obrázek C.3: *Pro 100 testovacích běhů vygenerujeme vždy rekurentní neuronovou síť vyvažující tyč na vozíku. Učení začíná při svislé poloze tyče, snažíme s dosáhnout 100000 kroků vyvažování. Graf zobrazuje průměrný počet kroků vyvažování tyče, jestliže natrénovaná síť započne vyvažování s již vychýlenou tyčí.*

C.4 Histogram počtu kroků vyvažování rekurentní sítě s $n_c = 15$ a 20% mutací



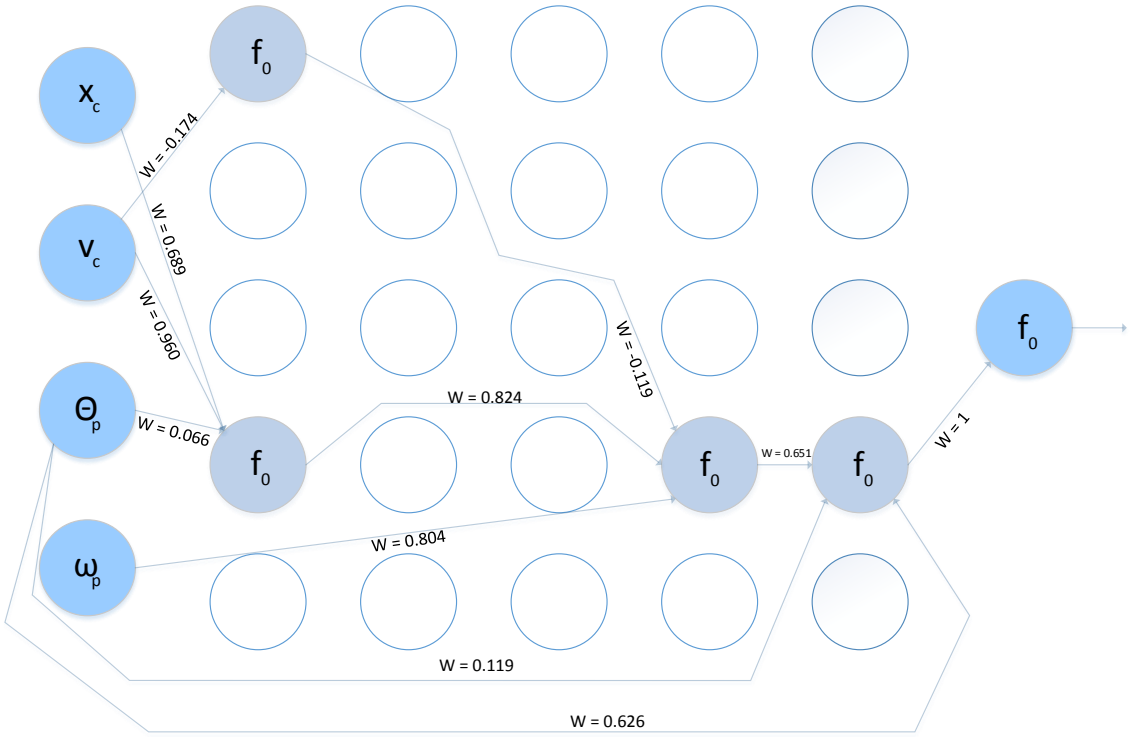
Obrázek C.4: *Pro 100 testovacích běhů vygenerujeme vždy rekurentní neuronovou síť vyvažující tyč na vozíku. Učení začíná při svislé poloze tyče, snažíme s dosáhnout 100000 kroků vyvažování. Graf zobrazuje průměrný počet kroků vyvažování tyče, jestliže natrénovaná síť započne vyvažování s již vychýlenou tyčí.*

C.5 Histogram počtu kroků vyvažování rekurentní sítě s $n_c = 5$ a 10% mutací



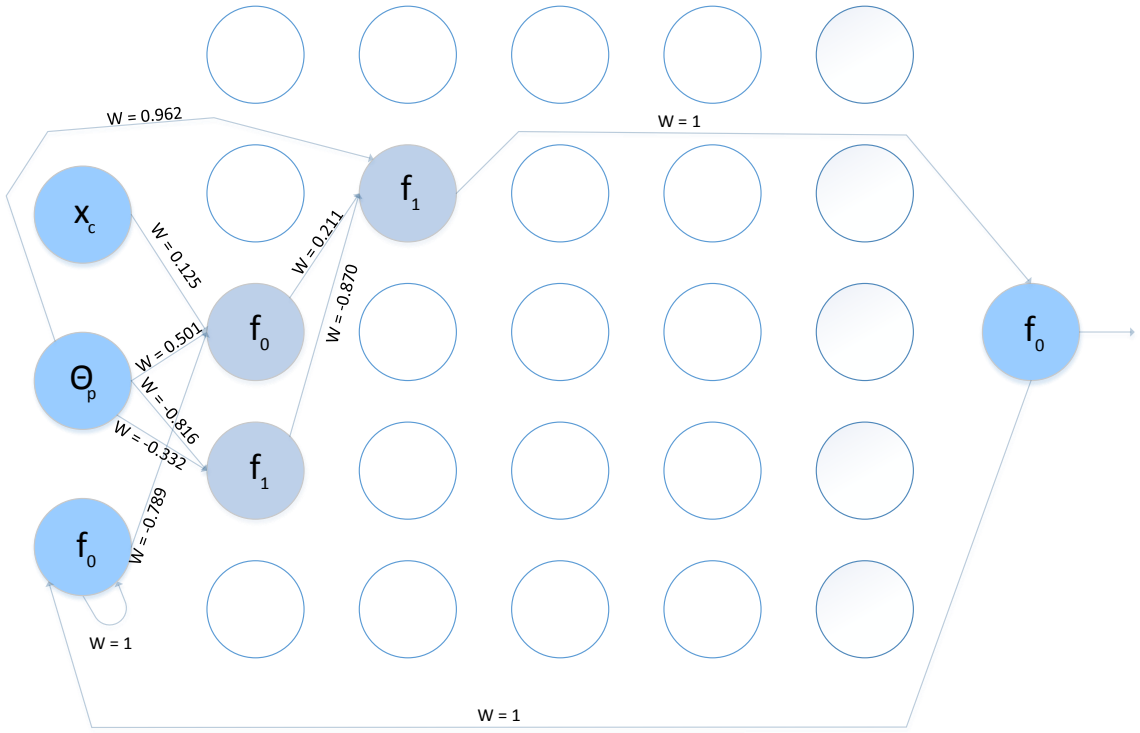
Obrázek C.5: *Pro 100 testovacích běhů vygenerujeme vždy rekurentní neuronovou síť vyvažující tyč na vozíku. Učení začíná při svislé poloze tyče, snažíme s dosáhnout 100000 kroků vyvažování. Graf zobrazuje průměrný počet kroků vyvažování tyče, jestliže natrénovaná síť započne vyvažování s již vychýlenou tyčí. Pro experimenty vybíráme nejrobustnější řešení z poslední generace. Ukazuje se, že to není přímo to nejspecializovanější.*

C.6 Konkrétní řešení dopředné sítě pro vyvažování tyče



Obrázek C.6: *Fenotyp pro síť vzniklou z chromozomu s parametry* $n_c = 5$, $n_r = 5$, $ES(1+9)$ *a* 10% *mutaci, kde* $f_0(x) = tanh(x)$. *Nezvýrazněné kruhy představují původní neaktivní uzly chromozomu pro představu o velikosti intronů v řešení. Síť reprezentuje optimální řešení, které vyvažuje tyč ve* 100000 *krocích a to včetně všech možných počátečních vychýlení.*

C.7 Konkrétní řešení rekurentní sítě pro vyvažování tyče



Obrázek C.7: *Fenotyp pro síť vzniklou z chromozomu s parametry* $n_c = 5$, $n_r = 5$, $ES(1+9)$ *a* 10% *mutaci, kde* $f_0(x) = tanh(x)$ *a* $f_1(x) = \frac{1}{1+e^{-x}}$. *Nezvýrazněné kruhy představují původní neaktivní jednotky chromozomu pro představu o velikosti intronů v řešení. Síť reprezentuje optimální řešení, které vyvažuje tyč ve* 100000 *krocích.*

Příloha D

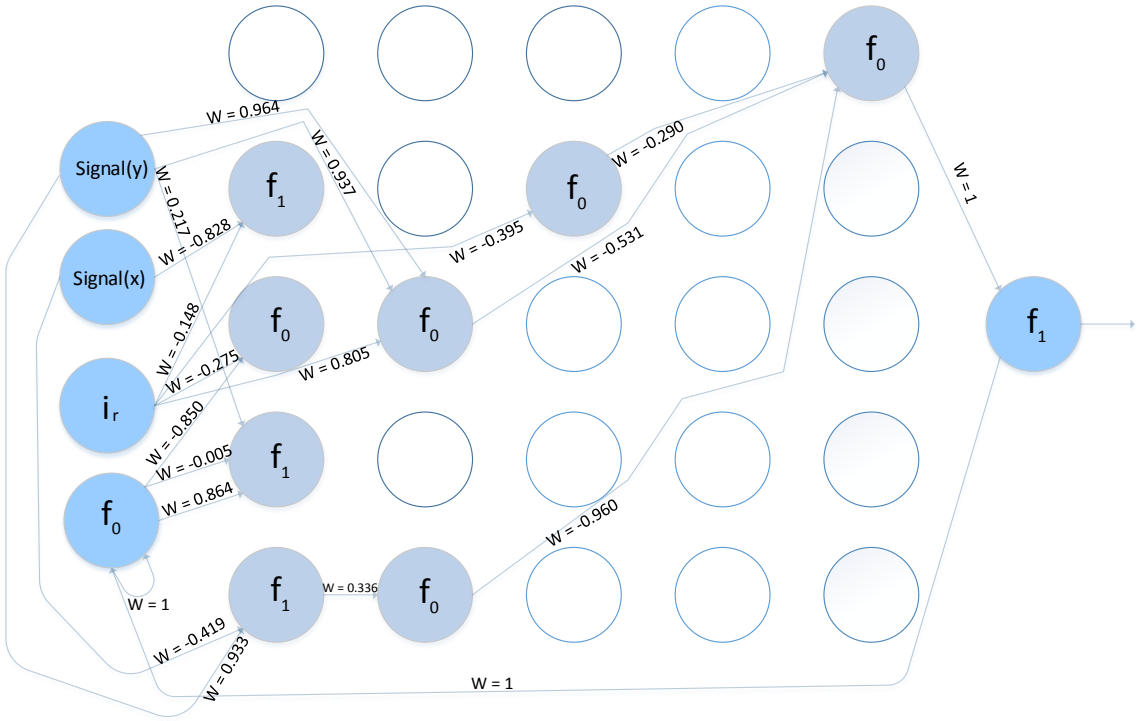
Příloha experimentů s bludištěm

D.1 Tabulka průměrné délky trajektorie v bludišti s překážkou

$n_c \times n_r$	10% mutace		20% mutace	
	$ES(1+9)$	$ES(5+5)$	$ES(1+9)$	$ES(5+5)$
5×5	14.31	15.83	14.81	16.06
15×15	14.15	15.35	15.46	16.59
50×1	14.38	15.23	15.20	16.23

Tabulka D.1: *Tabulka vyjadřuje průměrnou délku cesty v případě úspěšného průchodu bludištěm a vyhnutí se překážce po 11000 generacích evoluce. Nejkratší možná trasa byla délky* 13*. Bylo použito 100 testovacích běhů pro každou konfiguraci. Modře jsou znázorněny nejlepší řešení napříč mutacemi a evolučními strategiemi.*

D.2 Konkrétní řešení rekurentní sítě pro hledání uzavřené cesty v bludišti



Obrázek D.1: *Fenotyp pro síť vzniklou z chromozomu s parametry* $n_c = 5$, $n_r = 5$, $ES(1+9)$ *a* 10% *mutaci, kde* $f_0(x) = tanh(4x - 2)$ *a* $f_1(x) = \frac{1}{1+e^{-4x}}$. *Nezvýrazněné kruhy představují původní neaktivní jednotky chromozomu pro představu o velikosti intronů v řešení. Síť reprezentuje optimální řešení hledající nejkratší cestu.*

Příloha E

Obsah CD

code/	*Složka se zdrojovými kódy knihovny a experimentů*
lib/	*Zdrojové C++ kódy a uložené ukázkové zakódování*
*.cpp	
*.h	
cancerFNN.model	*Zakódování dopředné sítě pro detekci rakoviny*
mazeCycleRNN.model	*Zakódování rekurentní sítě hledající uzavřenou cestu v bludišti*
mazeBarrierRNN.model	*Zakódování rekurentní sítě hledající nejkratší cestu s vyhnutím se překážce*
poleBalancingRNN.model	*Zakódování rekurentní sítě pro vyvažování tyče*
poleBalancingFNN.model	*Zakódování dopředné sítě pro vyvažování tyče*
bestTopoFile2.model	*Modely mřížky s komponentami pro Developmental Network*
bestTopoFile1.model	
originalTopoFile.model	
Makefile	*Makefile pro knihovnu*
plot/	*Složka se scripty pro experimenty*
*.sh	
*.py	
*/	*Složky obsahující nezpracovaná experimentální data*
doc/	*Složka obsahující hierarchicky roztříděné dokumenty*
coevolution/	*Dokumenty týkající se koevoluce s CGP*
geneticProgramming/	*Dokumenty týkající se genetického programování*
neuralNetworks/	*Dokumenty týkající se neuronových sítí*
link	*Soubor se zajímavými webovými odkazy*
doxy/	*Doxy dokumentace*
doxylog.txt	*Šablona pro tvorbu doxygen dokumentace*
html/	
index.html	*Hlavní stránka předpřipravené dokumentace*
tex/	*Latexové soubory pro text diplomové práce*
*.tex	
literatura.bib	*Bibliografické citace*
Makefile	
/comps	*Prezentované texty*
/img	*Obrázky použité v práci*

README	
diplomaThesis.pdf	*Připravený text práce*